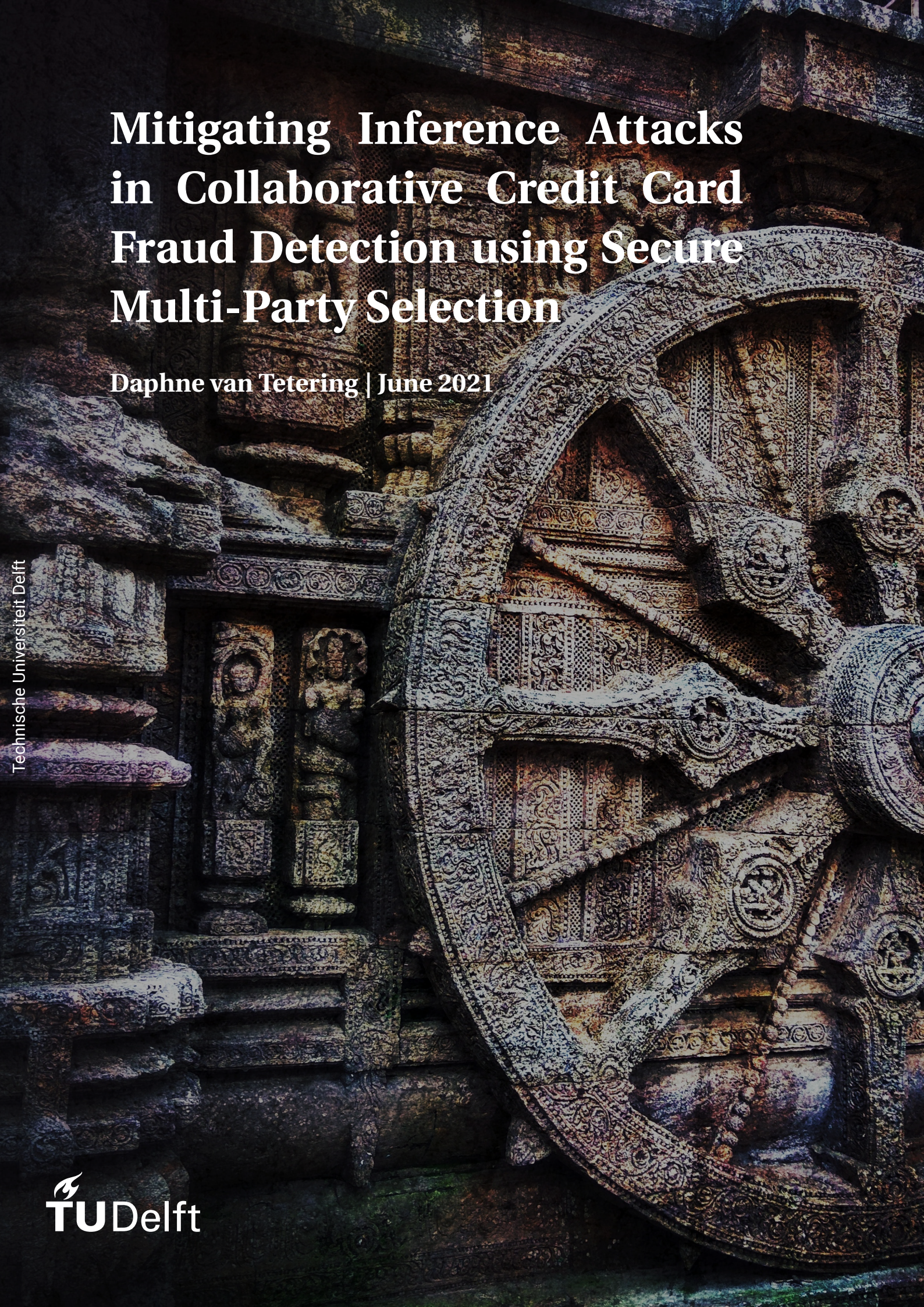


Mitigating Inference Attacks in Collaborative Credit Card Fraud Detection using Secure Multi-Party Selection

Daphne van Tetering | June 2021

Technische Universiteit Delft

TU Delft

Mitigating Inference Attacks in Collaborative Credit Card Fraud Detection using Secure Multi-Party Selection

Daphne van Tetering

to obtain the degree of Master of Science

at the *Delft University of Technology*,

to be defended publicly on Monday June 28, 2021 at 13:00.

Student number:	4375165	
Project duration:	September 7, 2020 – June 28 2021	
Thesis committee:	Prof. Dr. Ir. R.L. Lagendijk,	TU Delft
	Dr. M.F. Aniche,	TU Delft
	Dr. Z. Erkin,	TU Delft, supervisor
	T. Li MSc,	TU Delft, daily supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

The convenient service offered by credit cards and the technological advances in e-commerce have caused the number of online payment transactions to increase daily. With this rising number, the opportunity for fraudsters to obtain cardholder details via online credit card fraud has also increased. As a result, according to the European Central Bank, billions of Euros are lost due to credit card fraud every year. Since verifying all transactions by hand is infeasible, automated Fraud Detection Systems (FDSs) are needed. Currently, financial institutions create such systems by training machine learning algorithms on transaction data. However, the performance of these systems is obstructed due to a lack of positive (fraud) samples in the collected transaction data. To improve performance, an ideal solution would be to merge data of all institutions and to train an FDS on the resulting data set. However, privacy reasons concerning the sensitive customer information in this data, and security risks associated with transferring data, render this solution unrealistic. Therefore, the need rises for novel protocols that allow financial institutions to collaboratively train FDSs without sharing private data.

Previous research in the field of collaborative learning attempts to solve such problems by requiring participants to train local models, which are aggregated into a global model by a trusted central entity. Unfortunately, the vulnerability of these settings to inference attacks restricts their applicability. Inference attacks aim to extract additional secret knowledge from a model. These are especially powerful when performed by participants in a sequential setting, where participants train the same model one after the other following a given order. This is because in this setting participants have white-box access to the model itself and to the data used to train it. Naturally, these attacks are considered a breach of privacy and hinder collaboration. In this work, we propose a novel protocol leveraging secure multi-party computation techniques to prevent inference attacks in a sequential setting. To achieve this, we require participants to jointly determine a training order. While doing so, we ensure participants only receive information on whom to send their data to. This means participants are unaware of whose data they are receiving. With this work, we contribute a practical protocol that is robust against inference and timing attacks to facilitate privacy-preserving sequential collaborative learning. To the best of our knowledge, our work is the first to prevent inference attacks using a secure multi-party selection protocol with overhead of only a few seconds.

Preface

While pursuing both my Bachelor's and Master's degree in Delft, I never planned to graduate within the field of applied cryptography. In fact, only after finishing nearly three-quarters of my planned study program, I realized that the concept of privacy had not been addressed sufficiently during my studies. From a personal perspective, I felt the need to address this shortcoming and decided to focus my master thesis on privacy-preserving machine learning. This required me to take extra courses and thereby caused some delays in my study progress. However, looking back, I do not regret the decision I made at all. The protocol described in this thesis is the result of research on the cutting edge of data analytics and applied cryptography. Its goal is to facilitate multi-party machine learning, allowing parties to jointly train a predetermined machine learning model without the need to share sensitive data. By providing robustness to inference and timing attacks, while also keeping the computational and communicational overhead low, this protocol improves upon the existing state-in-the-art in this field.

My thesis project consisted of three phases, each taking approximately three months: literature research, protocol design, and protocol evaluation. During literature research, I read existing literature and used this to form my own vision on the existing state-of-the-art. As I wanted to focus on credit card fraud detection, I first read works conducting this in a non-private setting. Often, these works noted that due to a lack of positive samples, the obtained performance often was not satisfactory. The next step was to read about privacy-preserving machine learning and how this is used to facilitate collaborative machine learning to increase the number of positive samples during training. In order to identify research gaps, I looked into the strengths and weaknesses of these techniques, which brought my attention to inference attacks and motivated me to find ways to prevent these. Since my first approach was similar to multi-party shuffling, I read about such protocols determine if any of them could be readily applied. Since this was not the case, I decided to design my own protocol. This marked the start of the second phase, during which I designed several protocol versions. During this phase, I discovered that I needed a Joint Random Number Generation protocol. Since this protocol had not been published yet, I rewrote the paper and submitted it to SECRYPT. I am very proud of the fact that the paper was accepted. Finally, the last phase of my thesis started. This phase was focused on writing the thesis itself, but also on writing a paper. This paper is currently under submission at CANS.

I am very proud of the results achieved as part of my thesis so far. First and foremost, I want to thank my supervisor, Dr. Zekeriya Erkin, for his guidance throughout this process, both on an academic and personal level. During my thesis, I have learned valuable lessons that will definitely stay with me in the future. Secondly, I want to thank Tianyu Li, my daily supervisor, for the brainstorming sessions that helped me to turn my ideas into reality. I want to thank all other master and Ph.D. students of the research group for the countless coffee breaks with cake and laughter. Finally, I want to thank Dr. Maurício Aniche and Prof. Dr. Ir. Inald Lagendijk for being part of my graduation committee.

While this thesis is the result of the research I conducted over the past nine months, getting to this point was a much longer process. I am grateful for the friends and family that have supported me during this process. Especially, I want to thank my boyfriend Niek, my parents Jan and Sylvia, and my sister Leonie for their unconditional support and for encouraging me to pursue a degree at TU Delft. Finally, I want to thank my best friends for their support and for the countless memories we have made so far.

Contents

Figures

Tables

Protocols

1

Introduction

The convenient service offered by credit cards and the technological advances in e-commerce have caused the number of online payment transactions to increase daily. In 2019, this resulted in a total value of 162 billion Euro [4]. Almost a quarter of these transactions was conducted using credit transfers [4]. Because of this, the opportunity for fraudsters to obtain cardholder details via online credit card fraud has also increased. In 2018, card fraud volume increased by roughly 25% compared to 2017. At the same time, the total number of transactions only grew 11% [5]. In other words, card fraud volume grew faster than the total transaction volume. As a result, according to the European Central Bank, millions of Euros are lost due to credit card fraud every year [5].

Since verifying all transactions by hand is infeasible, automated Fraud Detection Systems (FDSs) are needed. Currently, financial institutions create such systems by training machine learning algorithms on transaction data. Unfortunately, performance of these systems is obstructed by a lack of positive samples in collected data [78]. This causes algorithms to fail to distinguish between fraudulent (positive) and benign (negative) samples. An ideal solution to improve this would be to merge data of all institutions and train an FDS on the resulting dataset, thereby increasing the number of positive samples available to the FDS. However, the sensitive information in this data and security risks associated with transferring it, render this solution unrealistic [93]. Moreover, laws as the General Data Protection Regulation in Europe and Personal Data Protection Act in Singapore prohibit data sharing [33, 98]. This motivates the need for novel algorithms and protocols that allow collaborative training of FDSs that improve current systems without the need to share sensitive data.

The research field focused on solving this problem is called *collaborative learning* [93]. It allows parties to jointly train a model without sharing data, either in a *parallel* or *sequential* setting [82]. Besides increasing the number of positive samples, bundling knowledge and resources also increases data diversity, thereby yielding higher quality models. Most previous work considers a parallel setting, in which parties train a model locally before sharing it with a central entity, e.g. a parameter server. The final model is then obtained using aggregation techniques, such as model averaging [65]. It is evident that when handling sensitive data the assumption of a trusted central entity does not provide sufficient security guarantees. Therefore, a sequential setting can be used in which parties train the model one after the other, following a predetermined order. Unfortunately, both settings expose parties to new vulnerabilities, so-called *inference attacks*. These attacks aim to extract knowledge from a model that was not included as a feature in the dataset or that holds for a subset of the data only. Using this information, attackers are able to identify whether a specific data sample was included in training data. From this, attacks can derive sensitive information, such as which bank someone is a customer of or, even more personally, what race someone is. Clearly, these constitute breaches of privacy. As we will discuss later in this work, these attacks are much more powerful in a sequential setting. Naturally, a vulnerability to these attacks hinders collabora-

tion. So far, the severe consequences of these attacks have been demonstrated, but mitigating them remains an open problem [1, 86]. Therefore, research focused on preventing inference attacks in a collaborative learning settings is required.

1.1. Automated Credit Card Fraud Detection

Over the last decades, machine learning has shown impressive results in solving pattern-recognition problems [41], such as natural language processing [107], speech processing [22] and image recognition [18]. While these results have increased the amount of research done in this area, they also caught the attention of big tech-companies such as Google, Amazon and Microsoft. This has caused them to establish their own cloud platform services, thereby enabling companies to perform data analysis without a need for in-house expertise and infrastructure [24, 42, 90].

On the one hand, these developments provide us with several benefits, such as tailored recommendations [111], voice controlled services [12], and disease detection [87]. On the other hand, for machine learning algorithms to produce correct results, vast amounts of data are required. This has caused the amount of data being collected to increase significantly [41]. For most applications, such as language recognition or image processing, large amounts of data can be collected by, for example, processing written text in news articles or reviews, or by processing video frames. However, for some applications, such as disease or fraud detection, general data collection methods do not suffice since the number of available positive samples is far too low [93]. For instance, in the case of credit card fraud detection, the amount of legitimate transactions far exceeds the amount of fraudulent transactions. Often, less than one percent of the samples is fraudulent [78, 79].

Several methods have been proposed to overcome the problem of class imbalance. These can be categorized in two main approaches: *sampling methods* and *cost-based methods* [49, 79]. Sampling methods balance the class distribution before training, either by removing samples from the majority class, so-called *undersampling*, or by adding more sampling from the minority class, so-called *oversampling*. An example of such an oversampling method is SMOTE, which generates synthetic training instances from the minority class by interpolation between such instances [10]. While these methods do change the balance within a dataset, they either leave out a data sample, decreasing the number of samples used in training, or increase the number of samples by using duplicate or synthetic ones. In other words, these methods do not add new, real-world samples to the dataset. Previous work has shown performance heavily depends on the used sampling approach [2].

Instead of addressing the imbalance between data samples by sampling, cost-based methods use different losses for the majority and minority class [79]. In credit card fraud detection, these losses often depend on the transaction amount and steer the algorithm into favoring false positives, samples incorrectly classified as fraud, over false negatives, missed fraud samples [79]. Since all false positives have to be examined separately, this could result in an unfeasible amount of transactions that have to be manually verified.

1.2. Parallel Collaborative Learning vs. Federated Learning

In a parallel collaborative learning setting, each party trains a model locally and shares gradient updates with a parameter server. At the parameter server, the final model is obtained using aggregation techniques such as model averaging [65]. This setting allows all parties to train the model simultaneously. While this setting is often used in previous work, the assumption of a non-malicious central entity does not provide sufficient security guarantees when handling sensitive data, as is the case in credit card fraud detection.

Besides collaborative learning, a different concept exists that allows parties to jointly establish a model without sharing data. This concept is called *federated learning*. Conceptually, this setting appears to be similar to parallel collaborative learning because both settings require participants to train a model locally. Additionally, in both settings, the global model is obtained through aggrega-

tion by a central entity. However, some important differences should be noted. First of all, federated learning involves hundreds or thousands participants that have relatively small datasets, while collaborative learning involves much fewer organizations that have large amounts of data, often with similar features [57]. Additionally, in a collaborative learning setting participants have much greater computation power. Because of these differences, federated learning is very suitable for edge computations on IoT devices [50], while collaborative learning is suitable when several organizations, such as banks or hospitals, desire to work together [118].

1.3. Inference Attacks on Collaborative Learning

Even though sequential collaborative learning allows joint training of a model without sharing data and removes the need for a central entity, it also exposes parties to new vulnerabilities: so-called *inference attacks*. During these attacks, an adversary tries to extract additional secret knowledge from a model [47]. In a *membership* inference attack, the adversary tries to infer whether a specific data sample was used to train the model. In a *property* inference attack, it tries to infer properties that were not included as a feature or only hold for a subset of the training data.

To perform a membership inference attack on a *target model*, two models are trained: a *shadow model* and an *attack model*. The shadow model is trained on data similar to the data used to train the target model to ensure that predictions made by the shadow model are similar to those of the target model. To train the attack model, we use predictions obtained by the shadow model and provide these with a ground truth label, indicating whether the used sample was originally included in the training set. This highlights the importance of the shadow model: since we have no knowledge of which samples were used to train the target model, we need a correctly trained shadow model to provide ground truth labels for the attack model [97]. Property inference attacks are performed in a very similar manner: instead of using data similar to the data used to train the target model, the adversary needs data that is labeled according to the property it wants to infer.

Inference attacks can be executed in two settings: a *white-box* and a *black-box* setting. In a white-box setting, attacks are performed by *insiders*, e.g. parties in collaborative learning. In a black-box setting, attacks are performed by *outsiders* [96]. Since parties in a white-box setting have access to truly positive samples to train the shadow model, these attacks are considered most powerful [103]. Therefore, in this work, we focus on attacks in a white-box setting.

1.4. Inference Attack Mitigation Techniques

Naturally, a vulnerability to inference attacks hinders collaboration. Unfortunately, while research has demonstrated the severe consequences of these attacks, mitigating or preventing them in a collaborative learning setting remains an open problem [82, 103].

Often, *record-level* and *participant-level differential privacy* are suggested as mitigation techniques. Record-level differential privacy is applied by adding noise to individual training samples, thereby obfuscating their original values [8, 113]. This method protects against membership inference attacks, but does not prevent property inference attacks. Additionally, Melis et al. and Shokri et al. show that to achieve sufficient privacy protection large amounts of noise have to be applied, thereby risking degradation of model performance [68, 97]. Participant-level differential privacy adds noise based on all contributions from users specifically, thereby ensuring analysis on the resulting dataset does not change when all entries from one user are added or removed [67, 72]. While this does protect against property inference attacks, research has shown that a large number of users, in the order of thousands, is required to ensure the joint model still reaches convergence [68]. As explained earlier, this is not the case in collaborative learning, which is the setting considered in this work.

1.5. Research Goal

In our research, we set out to design a privacy-preserving collaborative fraud detection system, that allows us to increase the number of positive samples available during training. We aim to do so without requiring participants to share their data, neither directly nor in an encrypted format, and without relying on a central entity. While these assumptions are often made in existing works, these provide insufficient privacy guarantees when handling sensitive transaction data. At the same time, our goal is to require minimal computation and communication overhead, while also providing robustness against inference attacks. Altogether, our research question is as follows:

How can we design a privacy-preserving collaborative learning system that increases the number of positive samples available during training, while introducing only minimal computation and communication overhead and providing robustness against inference attacks?

1.6. Contributions

Until now, related studies have shown the vulnerabilities of collaborative learning settings to inference attacks performed by both insiders and outsiders. While studies reason about possible attack mitigation strategies, recent work has shown that proposed strategies often fail to succeed in a sequential collaborative learning setting: either protection is aimed at one attack specifically or requires amounts of noise of magnitudes that degrade model performance. In this work, we focus on preventing both attack types at once, without impacting model performance.

Our contributions are as follows:

1. Two protocols for efficient joint random number generation for secure multi-party computation.

2. A privacy-preserving multi-party selection protocol for collaborative learning.

Altogether, this has resulted in the following publications.

Efficient Joint Random Number Generation for Secure Multi-Party Computation
Erwin Hoogerwerf, Daphne van Tetering, Aslı Bay, Zekeriya Erkin
Accepted at the International Conference on Security & Cryptography 2021 (SECRYPT).

In this work, we describe two protocols for efficient joint random number generation for secure multi-party computation. The first protocol relies on bit-wise sharing of individually generated random numbers. By requiring participants to share only one bit per round with a changing lead entity, this protocol minimizes the amount of information leaked to an adversary. The second protocol requires a single broadcast and relies on the security of the Diffie-Hellman key exchange. This protocol requires only one round to establish jointly generated numbers with a zero-sum. This work is included in Chapter 4 of this thesis.

Mitigating Inference Attacks in Collaborative Learning Using Multi-Party Selection.
Daphne van Tetering, Tianyu Li, Zekeriya Erkin
Under review at the International Conference on Cryptology and Network Security 2021 (CANS).

In this work, we outline the multi-party selection protocol that was the result of this thesis. It allows parties to jointly select a permutation without revealing it. To achieve this, parties are divided into two groups: *leaders* and *participants*. For security, we assume leaders to be *malicious* and participants to be *semi-honest*. To jointly select a permutation, we leverage secure multi-party computation techniques such as *distributed decryption*, ensuring a ciphertext can only be decrypted if all parties contribute, and *joint random number generation* to mask individual values [54, 115]. To evaluate our work, we conduct a security and complexity analysis. To show it is practical, we include a performance analysis. Results show that our protocol requires an overhead of only a few

seconds for settings with up to nine participants. To our knowledge, our work is the first to leverage secure multi-party computation techniques with minimal overhead to provide privacy and robustness against inference attacks in a sequential collaborative learning setting. It should be noted that even though we present it in a credit card fraud detection setting, our work is versatile and can be used in any setting that benefits from sequential collaborative learning, such as disease or telecommunications fraud detection [16, 55, 93].

1.7. Outline

This thesis is structured as follows. First, we discuss preliminaries and related work in Chapter 2 and Chapter 3. Then, we discuss two new joint random number generation protocols in Chapter 4 and a protocol to jointly select a training for collaborative learning in Chapter 5. To analyze the protocol for collaborative learning, we include a security, complexity and performance analysis in Chapter 6. Finally, we conclude the thesis in Chapter 7.

2

Preliminaries

In this chapter, we provide building blocks used in this work. First, we discuss the distributed version of the exponential ElGamal cryptosystem and how we apply commitment schemes. Next, we discuss our use of majority voting and zero-knowledge proofs. Finally, we discuss the assumption of anonymous communication channels.

2.1. Modulus of the Sum

In our protocol, the training order is selected collaboratively by jointly selecting a permutation from a permutation table held by the protocol leaders. To ensure this selection always lies within the table range $(n! - 1)$, where n denotes the number of participants, we ensure that the sum v of all submitted values v_i lies within this range. To achieve this, we use the modulus of the sum as defined by Sweeney and Shamos in [102]:

$$v = (\sum v_i) \mod n! \,. \tag{2.1}$$

2.2. Homomorphic Encryption

When using encryption, ciphertexts have to be decrypted before operations such as addition and multiplication can be performed on them. *Homomorphic encryption*, in contrast, allows these operations to be performed on ciphertexts directly. In our work, we use homomorphic encryption to ensure that individual values v_i as submitted by participants remain secret, while also allowing the aggregating party to obtain a correct result. Additionally, homomorphic addition of ciphertexts allows us to decrease the number of decryption operations to only one, since we are only required to decrypt the aggregated ciphertext.

To prevent leaders from decrypting the aggregated ciphertext alone, we use a *threshold* homomorphic encryption scheme. In our case, we set the threshold to l, the number of leaders in the protocol. This ensures a ciphertext can only decrypted when all leaders take part. Two threshold homomorphic cryptosystems were considered: *distributed Paillier* and *distributed ElGamal*.

Paillier cryptosystem

The Paillier cryptosystem requires an RSA modulus $N = p \cdot q$, for which p and q are two large, independent, randomly generated large primes. Additionally, for these it holds that $\gcd(pq, (p-1)(q-1)) = 1$. Additionally, it requires a public generator g. Using these, the Paillier cryptosystem is defined as follows:

Key Generation. Each public key pk_{p_i} is given by (n, g), each secret key is given by (λ, μ) for which $\lambda = \text{lcm}(p-1, q-1)$ and $\mu = L(g^\lambda \pmod{n^2})^{-1}$, with $L(x) = \frac{x-1}{n}$.

Encryption. Using fresh randomness $r \in \mathbb{Z}_n^*$, encryption of a message m is performed as:

$$E(m,r) = g^m \cdot r^n \pmod{n^2} \tag{2.2}$$

Decryption. Decryption of a ciphertext $E(m,r)$ is done as follows:

$$m = (L(c^\lambda \pmod{n^2}) \cdot \mu) \pmod{n} \tag{2.3}$$

Homomorphism. The Paillier cryptosystem is additively homomorphic. This means that multiplying two ciphertexts results in addition of the two plaintexts:

$$D(E(m_1) \cdot E(m_2)) = m_1 + m_2 \tag{2.4}$$

$$\begin{aligned} E(m_1) \cdot E(m_2) &= (g^{m_1} \cdot r_1^n) \cdot (g^{m_2} \cdot r_2^n) && (2.5)\\ &= g^{m_1+m_2} \cdot (r_1^n \cdot r_2^n) && (2.6)\\ &= E(m_1 + m_2)\,. && (2.7) \end{aligned}$$

Additionally, additive homomorphism ensures that exponentiation of a ciphertext with a plaintext message corresponds to a multiplication of the two in the plaintext domain:

$$D(E(m_1)^{m_2}) = m_1 \cdot m_2 \tag{2.8}$$

This can be verified as follows:

$$\begin{aligned} E(m_1)^{m_2} &= (g^{m_1} \cdot r^n)^{m_2} && (2.9)\\ &= g^{m_1 \cdot m_2} \cdot r^{n \cdot m_2} && (2.10)\\ &= E(m_1 \cdot m_2)\,. && (2.11) \end{aligned}$$

In this work, we require a distributed cryptosystem. To establish a distributed homomorphic Paillier cryptosystem, all public parameters have to be determined jointly. Among these is the RSA modulus n, consisting of two secret large primes p and q. Therefore, in order to determine n, two large primes p and q have to be determined jointly as well. Since both primes have to be kept secret, jointly generating or allowing a (trusted) dealer to generate them would require zero-knowledge proofs to prove both numbers are generated correctly without revealing them [13].

Exponential ElGamal Cryptosystem

Contrary to the Paillier cryptosystem, all parameters of the ElGamal cryptosystem are publicly known, among which large primes p and q, and the public generator g. Since the ElGamal cryptosystem is multiplicatively homomorphic, we encrypt messages as g^m instead of encrypting m directly to allow additively homomorphic aggregation of ciphertexts. This cryptosystem is also known as the Exponential ElGamal cryptosystem. In this work, we use the same cryptosystem as used in [115]. It requires a cyclic group G with prime order q and public generator g to be agreed on by all participants and is defined as follows:

Key Generation. Each party generates its secret key $sk_{p_i} = x_i \leftarrow [0,\dots,q-1]$ and its public key $pk_{p_i} = g^{x_i} \pmod{p}$. Using the generated keys pk_{p_i}, a shared public key is created as follows:

$$PK = \prod_{i=1}^{\ell} pk_{p_i} = g^{x_1+\dots+x_\ell}\,. \tag{2.12}$$

To allow joint decryption, all messages are encrypted using the shared public key PK.

Encryption. When using exponential ElGamal, we encrypt a message $m \in G$ as g^m instead of m directly. Using fresh randomness $r \in \mathbb{Z}_q^*$, this is done as follows:

$$\begin{aligned} c_1 &= g^r \\ c_2 &= g^m \cdot PK^r \end{aligned} \tag{2.13}$$

The resulting ciphertext is given by $E(m) = (c_1, c_2)$.
Shared Decryption. To perform decryption, each party partially decrypts the ciphertext and broadcasts the result. These are combined to retrieve the correct plaintext message. For a ciphertext $E(m) = (c_1, c_2)$, this requires the following steps:

- Each party computes ${c_1}^{sk_{A_i}}$ and broadcasts this together with a commitment $H({c_1}^{sk_{A_i}})$.
- Each party checks whether all received partial decryptions ${c_1}^{sk_{A_i}}$ and corresponding commitments $H({c_1}^{sk_{A_i}})$ match.
- If this is the case, each party computes

$$\frac{c_2}{\prod_{i=1}^{n} {c_1}^{sk_{p_i}}} = \frac{c_2}{{c_1}^{sk_{p_1}+\cdots+sk_{pn}}} = g^m. \tag{2.14}$$

- To retrieve m, each party computes the discrete log.

Homomorphism. Similar to the distributed Paillier cryptosystem, the Exponential ElGamal cryptosystem is additively homomorphic. This means the following relations hold:

$$D(E(m_1) \cdot E(m_2)) = m_1 + m_2 \tag{2.15}$$
$$D(E(m_1)^{m_2}) = m_1 \cdot m_2 \tag{2.16}$$

The first relation can be verified as follows:

$$\begin{aligned} E(m_1) \cdot E(m_2) &= (g^{r_1}, g^{m_1} \cdot PK^{r_1}) \cdot (g^{r_2}, g^{m_2} \cdot PK^{r_2}) \\ &= (g^{r_1+r_2}, g^{m_1} \cdot g^{m_2} \cdot PK^{r_1+r_2}) \\ &= (g^{r_1+r_2}, g^{m_1+m_2} \cdot PK^{r_1+r_2}) \\ &= E(m_1 + m_2)\,. \end{aligned} \tag{2.17}$$

The second one is verified as follows:

$$\begin{aligned} E(m_1)^{m_2} &= (g^{r_1}, g^{m_1} \cdot PK^{r_1})^{m_2} \\ &= (g^{r_1 \cdot m_2}, g^{m_1 \cdot m_2} \cdot PK^{r_1 \cdot m_2}) \\ &= E(m_1 \cdot m_2)\,. \end{aligned} \tag{2.18}$$

2.3. Commitment schemes

Suppose Alice and Bob are both in different places but want to flip a coin to determine who will pay for groceries. They decide that Alice should first take a guess, after which Bob will perform the coin flipping. While this might seem like a good idea, Bob is given the opportunity to influence the outcome of the coin toss by simply telling Alice the coin flipped in his favor. However, requiring Bob to flip the coin before having Alice announce her decision will allow Alice to cheat. Therefore, some method is needed to allow both Bob and Alice to make their decision before the coin flip, while also ensuring their decision cannot be changed once the coin is being flipped. To achieve this, commitment schemes are used [25]. These schemes allow Alice to send Bob a commitment with her decision, without revealing her decision beforehand. Then, after the coin has been flipped, Alice will send Bob additional information to reveal her decision in her commitment. This can be summarized below as a simplified commitment scheme [25]:

Simplified commitment scheme

Prover:

- Alice sets x to be heads or tails and sends the commitment $y = c(x)$ to Bob.
- Bob flips the coin and announces the result to Alice.
- Alice reveals her decision x to Bob.
- Bob verifies whether $y = c(x)$.

To ensure correctness of the commitment scheme, the commitment c must adhere to two requirements [25]:

- *Hiding* - Ensuring that Bob is unable to derive x from $c(x)$.
- *Binding* - Ensuring that Alice cannot reveal a different $x' \neq x$ for which $c(x) = c(x')$.

These requirements ensure that Alice cannot change her mind after sending commitment and that Bob can only determine Alice's choice after receiving x from Alice.

A commitment scheme can be based on different cryptographic primitives, such as cryptographic hash functions or the discrete log problem. Depending on the primitive, these binding and hiding properties are either computationally or information-theoretically secure. This means that the security of the commitment scheme depends on the computing power that we give an adversary that is tasked with breaking these properties. In other words, when we say that a commitment scheme is computationally binding, this means the binding property is guaranteed in situations in which an adversary is given at most polynomially bounded computing power. The same holds for a commitment scheme that is computationally hiding. When we say that a commitment scheme is information-theoretically hiding, this means that the binding property is guaranteed even in situations in which an adversary is given infinite computing power. The same holds for a commitment scheme that is information-theoretically hiding. With this, it is important to note that, by definition, no commitment schemes can exist that are both information-theoretically hiding and information-theoretically binding at the same time. This can be explained as follows: suppose we have a commitment $c(x,r)$ that is both information-theoretically hiding and information-theoretically binding. Then, since it is information-theoretically hiding, there exist values x' and r' such that $c(x,r) = c(x',r')$, otherwise the hiding property is broken. However, an infinitely powerful adversary can use these values to break the binding property. Because of this, a commitment scheme can never be information-theoretically hiding and binding at the same time.

Often, these two properties are explained using security games [23]. These security games are used to describe the interaction between two opposing parties, often an adversary and a challenger. Here, the adversary is the one tasked with breaking the scheme or property at hand, while the challenger is responsible for providing the adversary with the information it is allowed to use while trying to win the game. For the binding and hiding properties, the following security games are defined, as discussed in [99]:

Definition 1 *A commitment scheme is said to be information-theoretically (resp. computationally) binding if no infinitely powerful (resp. computationally bounded) adversary can win the following game.*

- *The adversary outputs values $x \in \mathbb{P}$ and $r \in \mathbb{R}$.*
- *The adversary must then output a value $x' \neq x$ and a value $r' \in \mathbb{R}$ such that $C(x,r) = C(x',r')$.*

Definition 2 *A commitment scheme is said to be information-theoretically (resp. computationally) hiding if no infinitely powerful (resp. computationally bounded) adversary can win the following game.*

- *The adversary outputs two messages x_0 and x_1 of equal length.*
- *The challenger generates $r^* \in \mathbb{R}$ at random and a random bit $b \in \{0,1\}$*
- *The challenger computes $c^* = C(x_b, r^*)$ and sends c^* to the adversary.*
- *The adversary's goal is to determine the value of the bit b.*

In this work, we use commitment schemes based on cryptographic hash function to prevent parties involved in decryption from changing the computed partial decryption value after sharing it with other parties. Given a cryptographic hash function H and randomness r, a commitment on m is computed as:

$$c = H(r||m) \tag{2.19}$$

In order to satisfy the binding and hiding properties, a collision resistant hash function has to be used. In Chapter 5, we discuss the used hash function and how this satisfies the binding and hiding requirements.

2.4. Majority Voting

Majority voting is used by each party to determine who to send their data to. In majority voting, parties vote once for one of n candidates, the party receiving the most votes wins. To represent votes, we use punch-hole vector ballots as presented in [51]. Each ballot consists of n options $C_n[i]$. If a party wants to vote for candidate i, it selects option $C_i[i]$ to be an encryption of one: $C_i[1]$. All other options will be an encryption of zero. To allow additively homomorphic tallying of votes, the exponential ElGamal encryption discussed previously is used.

2.5. Zero-Knowledge Proofs

Zero-knowledge proofs are cryptographic primitives that allow parties to prove knowledge of certain information, without revealing the information itself [25]. Often, two parties are involved. The first one is the Prover, who is required to convince the Verifier that she has knowledge of information. The second party is the Verifier, based on statements received from the Prover, she determines whether she is convinced about the fact that the Prover indeed knows the information she is claiming to know. To achieve this, often a series of interactions between the Prover and Verifier is used. These interactions often contain messages that include deterministic values. Then, based on combining the messages, some equality holds. When this is the case, this means that all verifications conducted by the Verifier are passed and that the proof is accepted. In other words, following the work by Feige et al., these messages include a predicate P for which the Prover is required to show that it knows the statement S to satisfy the predicate P [34].

Following the same work, a zero-knowledge proof must satisfy three requirements [34]:

- *Completeness* - Ensuring that when the statement S is true, an honest Verifier will indeed accept the proof.
- *Soundness* - Ensuring it is not possible to mislead an honest Verifier to accept a proof with a false statement S.
- *Zero-Knowledge* - Ensuring the Verifier will learn nothing about statement S except for its validity.

Fiat-Shamir Heuristic

An interactive zero-knowledge proof can be converted to a non-interactive one using the Fiat-Shamir heuristic [35]. This is done by taking the original interactive zero-knowledge proof and creating a signature from it. After doing so, the Prover shares the signature with the Verifier.

In this work, we use a non-interactive ZKP as defined by Yang et al. [115]. Specifically, this proof is suited to allow parties to prove ciphertext is encrypted from m without revealing m itself using the exponential ElGamal cryptosystem. This means that a cyclic group G, with prime order q and public generator g are available. Additionally, each party has a secret key x_i and public key $pk = g^{x_i}$. To prove that a ciphertext $E(m) = (c_1, c_2) = (g^r, g^m \cdot pk^r)$ was encrypted from m, it is necessary to prove that c_1 and $\frac{c_2}{g^m}$ have the same exponent. This is done using the following non-interactive zero-knowledge proof from [115]:

NIZK proof for Plaintext Knowledge

Prover:

- Generate random $r \in \mathbb{Z}_q^*$
- Compute $E(m) = (c_1, c_2) = (g^r, g^m \cdot pk^r)$
- Generate random $t \in \mathbb{Z}_q^*$
- Compute $T_1 = g^t$ and $T_2 = pk^t$
- Compute $v = Hash(E(m)||T_1||T_2)$ and $s = r \cdot v + t$
- Send c_1, c_2, T_1, T_2, v and s to the Verifier

Verifier:

- Verify that $g^s = {c_1}^v \cdot T_1$
- Verify that $pk^s = (\frac{c_2}{g^m})^v \cdot T_2$

If both verifications pass, (c_1, c_2) is a valid encryption of m.
Correctness for the first verification is verified as follows:

$$\begin{aligned} g^s &= c_1^v \cdot T_1 \\ g^{r \cdot v + t} &= (g^r)^v \cdot g^t \\ g^{r \cdot v + t} &= g^{r \cdot v + t}. \end{aligned}$$

Correctness for the second verification is verified as follows:

$$\begin{aligned} pk^s &= (\frac{c_2}{g^m})^v \cdot T_2 \\ pk^{r \cdot v + t} &= (\frac{g^m \cdot pk^r}{g^m})^v \cdot pk^t \\ pk^{r \cdot v + t} &= (pk^r)^v \cdot pk^t \\ pk^{r \cdot v + t} &= pk^{r \cdot v + t}. \end{aligned}$$

2.6. Anonymous Communication Channels

As discussed, participants only receive information on who to send their data to next. In other words, participants have no knowledge of whose data they are receiving, thereby preventing the execution of inference attacks. To guarantee this, it is important to provide *sender anonymity* using *anonymous communication channels*. Sender anonymity ensures a particular message cannot be linked to any sender, but also that a particular sender cannot be linked to any message [85].

Over the last decades, several methods to achieve anonymous communication channels have been established. Following [85], these can be divided into several categories. The first category distinguished are *mixnet-based* schemes, these schemes use a set of servers that mix incoming messages in such a way that the connection between sender and receiver becomes indistinguishable, for example by batching messages and randomly permuting before sending them to the next server. Examples of mixnet-based schemes are Onion Routing and Tor [27, 40]. The next category distinguished are *DC-net* systems, based on the Dining Cryptographers problem [17]. Finally, the overview by Ren and Wu considers *network routing-based* techniques, routing messages among nodes in a network, and peer-to-peer networks, in which a messages' route is chosen among the peers in the network. Additionally, the work by [64] uses a trusted third party to establish sender anonymity.

In this work, we assume anonymous communication channels are available to all participants.

3

Collaborative Fraud Detection Systems

Collaborative fraud detection systems are needed to allow financial institutions to jointly train a model without sharing data. In this chapter, we review several collaborative learning systems and show how these systems are vulnerable to inference attacks. Next, we discuss what privacy-preservation mechanisms are required to allow privacy-preserving collaborative learning and to prevent inference attacks. Finally, we discuss how, instead, the use of random shuffling protocols could help us in providing unlinkability between data and its owner, thereby mitigating inference attack risk.

3.1. Collaborative Learning

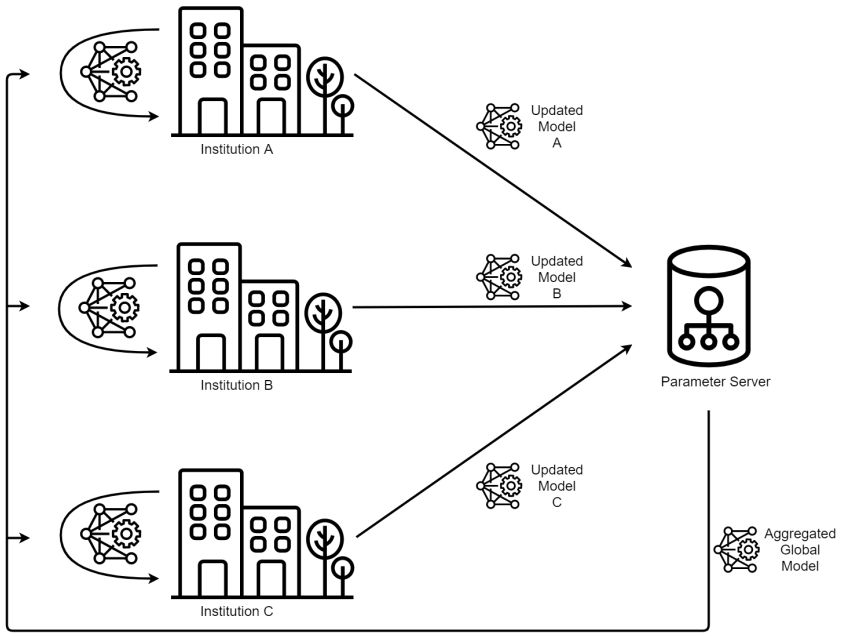
In this work, we assume a setting in which multiple parties want to work together to perform credit card fraud detection. By doing so, they are able to increase the number of samples available in training, while also adhering to data protection regulations such as the GDPR [33]. Two settings for collaborative learning exist: a *parallel* one, shown in Figure 3.1a, in which models are trained locally before being merged into a global one, and a *sequential* one, shown in Figure 3.1b, in which parties train the global model one after the other following a predetermined order [82]. In this section, we first discuss works that employ these two settings. Then, we discuss the vulnerability of these settings to inference attacks and the privacy-preserving mechanisms that are used to mitigate the consequences of these attacks.

A note on Federated Learning

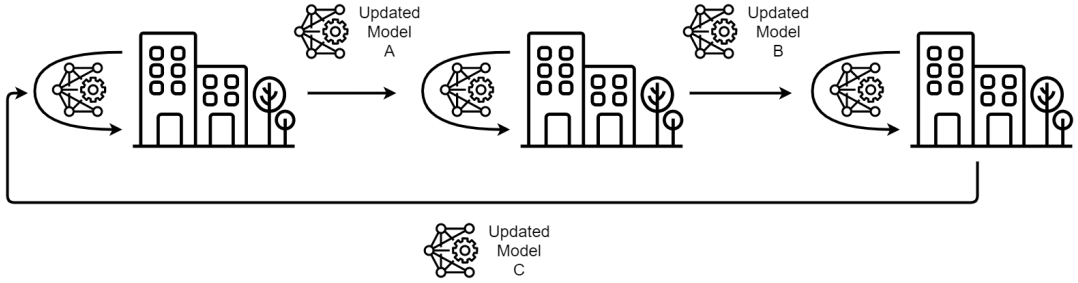
As explained in the introduction, parallel collaborative learning setting and federated learning are very similar. In this work, we clearly distinguish between the two settings. In related work, however, these terms are sometimes used interchangeably. As we will see in this section, some works consider a setting in which several institutions perform collaborative disease or credit card fraud detection, while calling this a federated setting. Therefore, when reviewing related work we will use these terms interchangeably. However, in other chapters, when we talk about institutions collaboratively detecting fraud, we refer to collaborative learning. When we refer to federated learning, we are talking about a setting as defined in the introduction.

3.1.1. Parallel Collaborative Learning

In this section, we discuss works that employ parallel collaborative learning in a non-private setting. In [114], the federated learning setting is used for credit card fraud detection. To balance the dataset, the minority class is over-sampled using SMOTE [10]. To simulate a credit card fraud detection environment, a public dataset containing transactions made by European cardholders in September 2013 is used. Due to the sensitive nature of transaction data, this dataset contains features that are the result of a PCA transformation. Originally, the dataset contains 0.172% fraudulent



(a) Parallel collaborative learning.



(b) Sequential collaborative learning.

Figure 3.1: Collaborative learning settings with three institutions.

transactions. Using SMOTE, several datasets are created with different ratios between fraudulent and non-fraudulent transactions. From this, it becomes visible that as the dataset becomes more imbalanced, fraud detection performance decreases. It is important to note that in this work, no privacy measures are considered. As noted by the authors, what information can be extracted from the final model or from individual gradient updates submitted by banks is not considered. This is left for future work.

In [21], Project Adam is proposed, a system aiming to provide scalable training of deep learning models. To achieve this, both the dataset and the model are partitioned across several machines that communicate with a parameter server. In an empirical study, authors show that using this system, a high accuracy can be obtained. In this work, no privacy-preserving measures are considered.

3.1.2. Sequential Collaborative Learning

In this section, we discuss works that use a non-private sequential setting. Our literature review has shown that, so far, sequential collaborative learning has not received much attention.

In [92], federated learning is compared with two collaborative learning settings: *institutional incremental learning (IIL)* and *cyclic institutional incremental learning (CIIL)*, which is the same as sequential collaborative learning. The task at hand is to determine tumor volume from MRI scans of the brain. All settings are evaluated using different numbers of institutions, varying between 2 and 32. With this, it is important to note that the total number of data points is fixed: this means

that as the number of institutions increases, the number of data points per institution decreases. Additionally, a centralized setting is used to evaluated whether learning in federated and collaborative settings increases model performance. Results show that IIL performs poorly compared to FL and CIIL. Additionally, the authors observe that IIL and CIIL are not suitable when the number of institutions is large and the number of data points per institution is small.

In [93], the same experiment as in [92] is conducted. However, in the supplementary material of this work, the security and privacy concerns of training models collaboratively are addressed. Unfortunately, the risk of inference attacks, executed by outsiders or insiders, is specifically left for future work. The same holds for malicious tampering of the model by participants in collaborative learning, such as model or data poisoning attacks.

In [16], collaborative learning is used in a medical domain to determine whether images of the eye depict a healthy or diseased one. The authors compare performance obtained in four different settings: a centralized setting, an ensemble of models, in which no data sharing occurs and each model classifies a sample after which all decisions are merged into one, a single weight transfer, similar to the IIL setting from [92], and a cyclical weight transfer, similar to the CIIL setting from [92] and also similar to collaborative learning. During the experiment, the number of institutions is set to 4 and the number of data points per institution is 1500. From all collaborative settings, the highest accuracy on the test set was obtained using cyclical weight transfer. Comparing this to accuracy obtained in a centralized setting shows that it dropped by about 2%. This shows that collaborative training yields performance similar to a centralized setting. The authors also highlight the fact that even when models were transferred among institutions multiple times overfitting did not occur. Finally, results show that a higher frequency of weight transfer, e.g. more training cycles at each institution, yields better accuracy compared to a low frequency of weight transfer. With this, it should be noted that more weight transfer also increases communication costs and may result in an increase of the total training time. One limitation of the study, which holds for all studies discussed so far, is that the datasets of institutions are sampled from one large dataset. Therefore, variability between institutions is low. The authors note that future work should explore scenarios in which there is much more variability between data of institutions. Concluding, this study shows that in collaborative learning a performance comparable to centralized settings can be obtained, thereby removing the need of sharing patient data directly. Unfortunately, the paper does not consider data security and data privacy risks.

3.2. Attacks on Privacy in Collaborative Learning

So far, we have discussed settings that benefit from collaborative learning but do not take into account the privacy risks occurring when employing such a setting. In this section, we discuss how not doing so exposes participants to powerful inference attacks. These attacks aim to extract secret knowledge from a model [47, 68]. In a *membership* inference attack, the adversary tries to infer whether a specific data sample was used to train the model under attack. In a *property* inference attack, it tries to infer properties that were not included as a feature or only hold for a subset of the training data [68, 86]. Membership inference attacks were first introduced by Shokri et al. in [97]. This work introduces the concept of membership inference attacks and demonstrates how to execute them on several classifiers. An example of property inference is found in [68]. This work shows that in a gender classification task, the classifier can also perfectly determine whether the current sample is of an Asian person or not.

Inference attacks on a *target model* are performed by training a *shadow* and an *attack model* [97]. To ensure that predictions of the shadow model are similar to those of the target model, it is trained on a similar dataset as the target model was trained on. To train the attack model, labeled predictions obtained from the shadow model are used. This means that an accurately trained shadow model is important for attack success. An overview of the attack model is shown in Figure 3.2. In-

ference attacks can be performed by *insiders*, e.g. participants in collaborative learning that have access to real data, and *outsiders*, that only have access to gradient updates. Before discussing how to mitigate these attacks, we first discuss related work showing their severe consequences.

3.2.1. Membership Inference Attacks

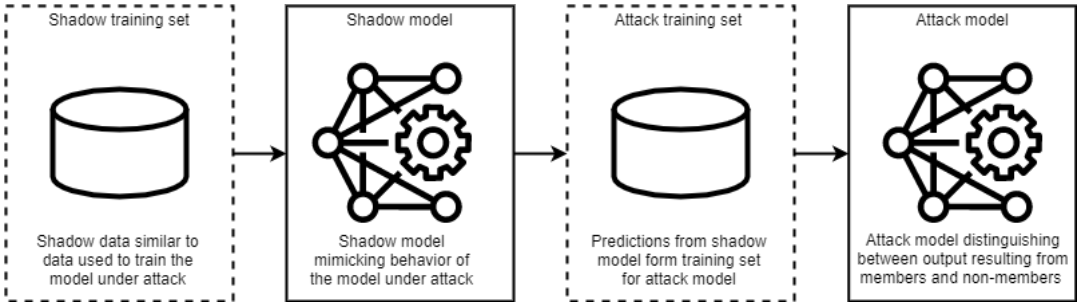


Figure 3.2: Membership inference attack setting.

In this section, we discuss works that focus solely on membership inference attacks.

In [103], Truex et al. compare the effectiveness of insider and outsider membership inference attacks by attacking decision trees in a parallel setting with three participants. Results show that even models with a minimal vulnerability to outsider attacks are significantly more vulnerable to insider attacks. For example, when attacking decision trees in a binary classification problem, inference precision jumps from 55.59% in an outsider attack to 73.26% in an insider attack. Inference accuracy also increases, from 59.89% in an outsider attack to 69.33% in an insider attack. Results also show that models that are highly vulnerable to outsider attacks have a similar vulnerability to insider inference attacks. For example, decision trees in an image classification problem allow an outsider inference precision of 83.94% and an insider inference precision of 82.02%. With their work, Truex et al. highlight the importance of preventing insider inference attacks.

In [88], Salem et al. show that membership inference attacks can be performed at a much lower cost than previously assumed. To show this, attacks are performed using three different adversaries to compare attack success. The first adversary is assumed to have a dataset that comes from the same distribution as the model under attack and to have black-box access to the model. The second adversary is also assumed to have black-box access to the model but does not have access to a dataset with the same distribution. This means that the second adversary has much lower attack capabilities compared to the first one. The third adversary has the least attack capabilities: it does not train a shadow model but uses only posteriors obtained by querying a data sample. When comparing performance based on precision and recall, it becomes visible that often the second adversary yields better performance than the first. This shows that membership inference attacks can also be performed without the need for a shadow model, thereby proving that membership inference attacks indeed require much less computation power than assumed so far. Because of this, the risk of membership inference attacks is much higher than previously assumed.

In [84], a membership inference attack is launched against a differentially private deep model. To assess the trade-off between privacy protection and model utility, several modes with different levels of privacy budgets ϵ are evaluated. These are compared to a baseline setting of 0.5, corresponding to a situation in which an adversary can only make random guesses. Results show that for $\epsilon \leq 2$ privacy is protected sufficiently since attack success is lower than the baseline. For higher values of ϵ, e.g. 4 or 8, privacy is not protected, meaning that membership inference yields good performance. Moreover, utility of the model is only moderate. The value used for ϵ denotes the trade-off between model performance and privacy protection: for smaller values of ϵ, privacy is protected better but model performance is often found to degrade more. For larger values of ϵ, model per-

formance is not affected but privacy is not protected either. Following this, it is no surprise that for values such as 4 or 8, privacy is not protected. Later in this chapter, we discuss differential privacy in more detail.

In [105], Truex et al. introduce a system to analyze and evaluate trained models for privacy and compliance. Using this system, they show that a model trained on a skewed dataset is much more vulnerable to inference attacks on the minority class compared to a model trained on a balanced dataset. This is explained by the fact that when a dataset is skewed, the model often overfits on the minority class, thereby increasing membership inference vulnerability. Additionally, the system is used to determine the usefulness of differential privacy for attack mitigation. This is done by evaluating differentially private models on several classification tasks and comparing their utility to non-private counterparts. Results show that as the problem gets more complex, e.g. contains more classes, the vulnerability of non-private models to attacks increases. However, results also show that when increasing complexity, the utility loss caused by the use of differential privacy increases with it. The same result is found when taking into account data skewness: as a dataset gets more imbalanced, utility loss due to the use of differential privacy gets worse.

3.2.2. Property Inference Attacks

In this section, we discuss works that also focus on property inference attacks. Often, these works focus on both membership and property inference attacks.

In the work by Melis et al., membership and property inference attacks are executed in settings with varying numbers of participants [68]. This number is varied between 4 and 30. To compare attack success the AUC score is used. This measure is used to evaluate how well a machine learning model is able to distinguish between the positive and negative class [11]. In other words, the higher the AUC score, the better a model is performing. Results show that for up to 12 participants, property inference attacks achieve reasonable performance, yielding an AUC score of 0.8. When increasing the number of participants further, performance degrades quickly. Moreover, results also show that the model is able to precisely infer when a participant with the target property occurs in the dataset. While emphasizing the importance of preventing inference attacks once more, Melis et al. also demonstrate that the consequences of these attacks are much more severe than they seem, since they allow attackers to precisely infer when a participant with the desired properties occurs during training as well.

In [1], privacy and security issues in federated learning are investigated. To do so, four phases in the federated learning setting are defined: initialization, local updates by participants, model aggregation by the central server, and convergence, indicating the trade-off between privacy and the number of rounds required for convergence. The authors consider the risk of both membership and property inference attacks during the local update phase, in which participants train the model locally before sharing it with the server. Unfortunately, the authors do not execute and evaluate attacks themselves, they merely argue that based on results obtained in other works, membership and property inference form a threat to privacy and security in federated learning and that therefore mitigation techniques have to be designed.

In [110], attacks on user-level privacy in federated learning are executed. In these attacks, specific user-dependent properties that distinguish users from others are used to attack one participant specifically. This is done by training a generative adversarial network on these properties, thereby enabling it to generate samples similar to the ones owned by the participant under attack. Using these generated samples, a discriminator model is trained to distinguish which user owns a given sample. The attack is performed in both a passive and an active setting. In a passive attack, the server is assumed to be honest-but-curious, meaning it only analyzes updates after aggregation. In an active attack, the server is malicious and allowed to analyze gradients received from clients separately. This means the server can decide to isolate one participant, by analyzing their gradients sep-

arately and by giving them a wrongfully aggregated model. Since this causes the isolated participant to train using gradients that were not averaged, this will cause them to reveal even more information about their model. To evaluate attack success, attacks are executed in several settings for different image classification tasks. Attack success is measured by comparing the generated samples to the ones originally in the dataset. Results show that with the attack an adversary can precisely recover original samples, further emphasizing the risk of user-level privacy attacks in federated learning. With these results, it is important to note that they are obtained using image data, while transaction data is numerical.

Similar to the previous work, the next work also shows how training data can be reconstructed from shared gradients using a deep model and its weights [119]. This is done by first generating dummy pairs of inputs and corresponding labels to obtain dummy gradients. This process is repeated by changing the dummy inputs until the difference between obtained gradients and real gradients becomes negligible. When this is achieved, the dummy inputs look similar to the original data samples. Attack success is evaluated on an image classification task, where the goal is to reconstruct original images, and on a masked language model task where a model is required to fill in blanks in a sentence, here the goal for the attack model is to reconstruct original sentences. Results show that after 500 iterations for image reconstruction and 30 iterations for sentence reconstruction the samples have been reconstructed successfully.

3.3. Privacy Preserving Collaborative Learning

So far, we have considered non-private settings that benefit from collaborative learning and the severe consequences of inference attacks in these settings. In this section, we discuss mechanisms used to preserve privacy and to prevent inference attacks in both parallel and sequential learning. In literature, four privacy-preserving methods are commonly used, these are: *homomorphic encryption*, *differential privacy*, *dropout*, and *regularization*.

3.3.1. Homomorphic Encryption

As discussed in Chapter 2, homomorphic encryption allows operations such as addition and multiplication to be performed on ciphertexts directly, without the need for decrypting them first. Three types of homomorphic encryption are distinguished: *partially* homomorphic encryption (PHE), *somewhat* homomorphic encryption (SWHE), and *fully* homomorphic encryption (FHE) [75].
Whether a cryptosystem is fully, partially, or somewhat homomorphic depends on the type of operations that are supported and on the number of times these operations are allowed to be performed on ciphertexts. When both addition and multiplication are allowed, a cryptosystem is fully homomorphic. When only one of the two is allowed, it is partially homomorphic. Finally, a cryptosystem is somewhat homomorphic when it supports both addition and multiplication, but the number of operations that can be performed while also ensuring the aggregated ciphertext also contains the correct value is limited. While fully homomorphic cryptosystems might seem most attractive due to their ability to support both addition and multiplication, it is important to note that these are computationally more intensive compared to partially and somewhat homomorphic cryptosystems [70, 75]. When taking into account works that use homomorphic encryption, it becomes clear that this technique is mostly used in a parallel collaborative learning setting where the final model is obtained through aggregation by a central entity.

The first work we discuss uses *somewhat homomorphic encryption* to propose a cloud service that allows training and prediction of an algorithm in the cloud [43]. It considers three types of parties: a data owner, a cloud service provider, and several content providers. In the system, content providers are devices or persons uploading data that belongs to data owners to the cloud service provider. The use of a somewhat homomorphic cryptosystem prevents leaking data to the cloud service provider, while allowing it to perform all required operations on encrypted data. It should be

noted that due to the use of somewhat homomorphic encryption, the number of iterations available for gradient descent is limited. This hinders the applicability of the cloud service since this means it can only be used to train model where the number of iterations is guaranteed to be within this limit. This conclusion is also drawn by Wang et al. [109].

In [62], Ma et al. propose a privacy-preserving and high-accurate outsourced disease predictor (PHPR), allowing parties to train a model using distributed medical data. To achieve this, each party shares its dataset with a central entity using *additively homomorphic Paillier encryption*. In other words, after encryption, all datasets are merged together to form one. Using this dataset, a random forest is constructed. To perform prediction, a party uploads an encryption of the sample it wants to obtain a prediction for. The central entity then returns the corresponding prediction. In this work, all parties and the central entity are considered honest-but-curious.

In [52], an *homomorphic encryption* scheme that allows the use of real numbers is used to perform classification using a logistic regression classifier. In the used setting, each party encrypts its data using the homomorphic encryption scheme, after which a logistic regression model is trained on a central server. To evaluate their approach, the authors train the model on several datasets containing data on medical problems such as myocardial infarcts and prostate cancer. Results show that when comparing the model trained on encrypted data to one trained on plaintext data, similar results are obtained. Often, the encrypted model yields accuracies that are only 2% less compared to the plaintext model. Additionally, results show that it takes approximately 116 minutes to obtain a trained model from the encrypted dataset. However, it is important to note that in this work a security setting of only 80 bits is considered. Therefore, it is reasonable to expect that when increasing the security parameter, the overhead caused by the use of homomorphic encryption will increase with this.

Similarly, in [56] logistic regression is performed on a cloud server by requiring parties to encrypt all features of all data points using *fully homomorphic encryption*, before sharing these with the server. To achieve one-way security, the encryption scheme is defined over a non-abelian ring. However, as noted by the authors, as the desired security parameter is increased, ciphertext length increases with it. How to make the encryption scheme more efficient and how to reduce the ciphertext expansion rate is left for future work.

In [58], two schemes are proposed to preserve data privacy. The first one uses a *multi-key fully homomorphic encryption* scheme, while the second one uses an encryption scheme with a *double decryption mechanism*. A multi-key encryption scheme means that one message can be encrypted using multiple keys [19]. When a scheme has a double decryption mechanism, this means that there are two decryption algorithms required to decrypt a ciphertext, instead of one. The first algorithm can be executed by all parties using their private key, while the second can only be performed by a master entity that possesses a master private key. The threat model considered is an honest-but-curious one: both the data owners and the server are considered to be honest-but-curious. In light of this threat model two security goals are defined: *data privacy*, stating that data should be kept secret even if a part of the data owners and server are corrupted, and *training model privacy*, stating that the training model can only be known to data owners. To ensure this, each data owner uploads its data in an encrypted manner, after which the cloud server returns encrypted parameters of a trained model. Similar to other works that upload encrypted data to the cloud, this results in additional computation and communication overhead. Reducing the communication and computation overhead of the scheme is denoted as future work.

The next work we discuss is by Zheng et al., in this work a new system, Helen, for *coopetitive learning* is proposed [118]. The term coopetitive learning is used to denote a setting in which competitive institutions, such as banks, want to collaborate. As a potential use case, the authors name money laundering detection. From this, it follows that the setting considered in this work is very similar to the one considered in our work: it involves a few organizations, often less than 10, that

have access to large amounts of data with a smaller number of features. The proposed system allows these organizations to jointly train a model without sharing data while protecting them against malicious attackers. Using *threshold partially homomorphic encryption*, zero-knowledge proofs, and a maliciously secure multi-party computation protocol, Helen ensures that as long as participants follow the protocol honestly, malicious participants are unable to learn anything about the honest participant's data other than the final result of the function. From this, it becomes clear that Helen prevents other participants from executing membership and property inference attacks. However, the authors specifically mention that ensuring the model does not leak information about the data is not of relevance for Helen. They merely reason how such techniques could easily be added to the system. Helen works as follows: instead of sharing data or gradients directly among participants, each participant is required to share an encrypted summary of its data with all others. Then, each participant runs the pre-determined model using the received summaries. In this phase, zero-knowledge proofs are used to prove that each participant uses valid summaries of its data. The final model is obtained by using threshold decryption. The performance of this system is measured on two datasets, one containing readings from a gas sensor, where the task is identify the gas type, and one containing songs, where the task is to predict a song's publishing year. The performance of Helen on both datasets is compared to a centralized model. From this, the authors conclude that Helen reaches similar performance as its baseline model. While this work shows that when using secure multi-party computation techniques reasonable performance can be obtained, it does not measure whether these techniques protect participants from malicious participants trying to infer information about the data they have. Additionally, since the system was not evaluated using transaction data, it is uncertain if using encrypted summaries of transaction data will allow banks to train a well-performing model.

In [61], *homomorphic ElGamal encryption* is used in a parallel setting. Different from previous work, encryption is done using a multi-key ElGamal cryptosystem. This means that to perform decryption, multiple keys are required. Similar to the works discussed before, the server obtains the final model through aggregation of the received parameters. In this work, the server is assumed to be malicious but also to be non-colluding. This means that the server might deviate from the protocol but that it will never collude with participants. To allow for this, aggregate signatures are used. These allow participants to verify the result received from the server.

In [9], a federated learning setting is considered. To obtain the global model, this work proposes a *secure aggregation* algorithm using encrypted secret shares. During the protocol, all parties are considered to be honest-but-curious. Using this scheme, the central entity responsible for aggregation only has knowledge about the aggregated model, but not about individual ones.

In [14], a privacy-preserving decision tree based on *homomorphic encryption* is used to perform fraud detection. It is tasked with categorizing data samples in one of four categories: safe, moderately risky, very risky and fraud. Results show that when performing classification on encrypted data, computation and communication overhead increases with an order of five. Additionally, this work underlines the trade-off between additional overhead, key length, and ciphertext length: since a longer key produces longer ciphertexts, increasing the key length, for example for security reasons, also increases classification time [14].

The final paper we discuss specifically addresses several defenses, among which the use of homomorphic encryption, to prevent attacks on user-level privacy in federated learning [100]. The first defense discussed is the use of an *encryption-based protocol*, such as secure aggregation or homomorphic encryption, thereby requiring the server to perform operations on encrypted data. While this is an effective defense, the authors also note that it increases computation complexity and that using this makes it hard to recognize malicious updates. As a second defense, the use of *trusted execution environments* is discussed. When placing the gradients in such an environment on the central server, their confidentiality and integrity can be guaranteed. Since in this work we consider

a sequential setting, this defense is not applicable. Finally, the authors propose *dynamic participation,* allowing users to drop in and drop out of training at any time. The authors reason that this will hurt the attacker's shadow model since periodic updates are required to keep it relevant. With this, the authors do note that this might hurt model performance. In our case, since participants are required to follow a predetermined order, dropping in and out of participants is not wanted.

3.3.2. Differential Privacy

The term *differential privacy* was first suggested by Dwork and indicates the fact that removing or adding an item to a database should not affect the outcome of any analysis performed on the database [29]. From this, it follows that there is no risk in joining the database. Formally, differential privacy is defined as [29]:

Definition 3 *A randomized function K gives ϵ-differential privacy if for all datasets D_1 and D_2 differing at most one element, and all $S \in Range(K)$: $Pr[K(D_1) \in S] \leq \exp(\epsilon) \times Pr[K(D_2) \in S]$.*

In this definition, the parameter ϵ denotes the level of privacy that is obtained. The lower the value for ϵ, the stronger the privacy guarantee. In practice, values of 0.01 or 0.1 are used [36]. The paramater ϵ is also referred to as the *privacy cost.* This is due to the *composability property* of differential privacy, which means that when multiple queries are executed the differential privacy bound is computed by multiplying ϵ with the number of queries executed [36].

When looking at this form of differential privacy from an adversary's perspective, it ensures that even when an adversary knows the whole dataset D except for one target element, he is still unable to infer whether the target element exists in D using the output of the randomized function K [109].

Two types of differential privacy can be distinguished: *record-level* differential privacy, used to denote differential privacy as discussed so far, and *participant-level* differential privacy as defined by McMahan et al. [66]. This form of differential privacy states that behavior of a model should not change when all data belonging to a specific user is removed or added from the database [66]. From this, it follows that an attacker is unable to distinguish between which data point belongs to which user. In participant-level differential privacy, the following adjacency relation, as given in [66], is used:

Definition 4 *User-adjacent datasets: Let D_1 and D_2 be two datasets of training examples, where each example is associated with a user. Then, D_1 and D_2 are adjacent if D_2 can be formed by adding or removing all of the examples associated with a single user index from D_1.*

This new definition and application of differential privacy is especially useful when one user contributes multiple examples to the dataset, as is the case in transaction data. Using this definition, removing or adding any specific user's data to the dataset does not have a perceptible impact on an analysis done on the dataset. Consequently, an adversary is unable to infer whether a specific user's data was used during training, regardless of the additional information it might have [66]. By definition, record-level differential privacy prevents membership inference attacks but does not protect against property inference. Participant-level differential privacy does prevent property inference attacks.

In [28], *differential privacy* is used in a federated setting to protect gradients from a server that aggregates parameters using weighted parameter sharing. In this work, differential privacy is ensured by using object function perturbation, which applies Laplacian noise to the objective function, and by using output function approximation, which injects noise into the coefficients of the objective function itself. The framework is evaluated in a setting with three participants, each holding a different portion of the dataset: the first party holds 40%, the second 30%, and the third only 10%. The remaining 20% is used as a test set. To determine the trade-off between performance and privacy protection, the parameter ϵ is varied from 0.1 to 3.2. Results show that for values of ϵ

between 0.1 and 1.0, neither perturbation function yields performance similar to their non-private counterparts. For values larger than 1.0, this is the case. However, as discussed these values do not yield sufficient privacy protection. Therefore, the results obtained in this work clearly underline the trade-off between privacy and performance.

In [76], an untrusted curator Charlie is used to create additive shares of the final classifier and to share these with all participants. To protect individual data points, *differential privacy* is used. Additionally, to prevent Charlie from seeing model parameters, *homomorphic encryption* is used. All parties are considered to be honest-but-curious. To evaluate the performance of the fraud detection system, an experiment using five participants and different data splits is conducted. Additionally, different values for ϵ are compared. Results show that when data is split evenly across all participants, values for ϵ can be relatively low while still yielding good results.

In [41], a privacy-enhanced multi-party deep learning framework is designed that defends against attacks of honest-but-curious participants and an honest-but-curious server. To achieve this, the authors combine *differential privacy* and *homomorphic encryption*. While doing so, the authors claim to also keep the used privacy budget ϵ low. To show this, the authors compare results obtained using an ϵ of 10.0 and an ϵ of 1.0, while also varying the number of participants. Results show that for 20 participants, only a high privacy budget yields similar performance as a centralized setting. Fortunately, as the number of participants increases, the performance obtained using a smaller budget also increases. However, it should be noted that even when increasing the number of participants to 60, a small privacy budget does not yield performance similar to a non-private centralized setting. Therefore, this work again underlines the trade-off between privacy protection and model performance when using differential privacy.

In [104], *differential privacy* and *homomorphic encryption* are combined to allow the use of differential privacy protection against inference attacks without introducing too much noise. The work assumes a federated setting with an honest-but-curious aggregator. Threshold Paillier homomorphic encryption is used to protect against colluding parties, ensuring that decryption can only be performed with the participation of at least t honest parties. Unfortunately, the system is evaluated using too large values for the parameter ϵ, such as two and eight. This means that even though the trained models perform well, privacy is not protected sufficiently.

In [109], a parameter server is used in a sequential setting to perform credit card fraud detection: once a bank is done training, it uploads the resulting gradients to the parameter server. Then, the parameter server forwards these gradients to the next bank. To protect the privacy of the datasets and gradients, *differential privacy* is used. To ensure that privacy protection is sufficient while also allowing the model to reach sufficient performance, the authors propose to only share large gradients with the server. Since the amount of noise that has to be added to ensure differential privacy is linearly dependent of the number of gradients that is shared with the server, the authors reason this allows them to reach sufficient performance with strong privacy protection. To evaluate whether this is the case, a real-world credit card transaction data set, contributed by an American bank, is used. It contains 78 million transactions from over two million accounts. All transactions were executed from January until August 2013. To illustrate the trade-off between performance and privacy and to determine whether privacy protection is indeed stronger compared to previous works, several values for the parameter ϵ are compared in a setting with five parties. From this, it becomes visible that for values of 1.0 and higher performance comparable to a non-private version is obtained. Unfortunately, for smaller values such as 0.01 or 0.1, performance still drops below the performance of the non-private version. Altogether, this work shows that by sharing less gradients with the server, performance similar to a non-private model can be achieved using stronger privacy guarantees compared to other works.

The work in [86] specifically outlines defenses against attacks on machine learning models. Some of these are aimed at preventing different types of attacks, such as model stealing or recon-

struction attacks. Here, we discuss the ones that are relevant for mitigating membership and property inference. First, the authors discuss the use of *differential privacy*. However, similar to [68], they note that introducing differential privacy on a participant-level requires a too large number of participants. The second method discussed is *prediction vector tampering*, which means the prediction vector is modified to only contain the top-k common classes. In this work, we consider privacy during training, thereby rendering prediction vector tampering methods unsuitable. Finally, *random flipping* of labels is discussed. While this might help, it is likely to also impact model performance. Unfortunately, the authors merely reason about how these defenses could mitigate attacks but do not evaluate them for success.

In [1], authors propose the use of *cryptographic techniques*, such as *homomorphic encryption*, and *differential privacy* to mitigate membership and property inference attacks during the training phase. Unfortunately, they do not evaluate these methods for effectiveness. The authors conclude their work by discussing future directions for federated learning. Interestingly, here the authors note the opportunities in so-called *incremental federated learning*, where clients train the same model on their own datasets, as is done in sequential collaborative learning. Additionally, the authors also note that *decentralized federated learning*, in other words, federated learning without a central entity, could pose interesting research avenues.

In [20], *differential privacy* is used to protect against membership inference attacks. Here, the authors focus on information that is leaked by sharing models trained on genomics data. Consequently, differential privacy is applied to the model gradients. To evaluate effectiveness, the authors first compare model performance for different privacy budgets. Results show that as the privacy budget decreases, model performance degrades rapidly. Secondly, the authors assess the effectiveness of differential privacy in preventing membership inference attacks. To do so, attacks are executed on models trained using various privacy budgets. Results show that when the privacy budget is smaller than 10.0, the target accuracy of the models under attack becomes unstable and decreases rapidly. Because of this, the authors reason a privacy budget ϵ of 10.0 gives the best trade-off between privacy and target accuracy.

In [119], three defense strategies are outlined against information leakage from gradients. The first one is the use of *noisy gradients*. This means that, similar to differential privacy, noise is added to the gradients. To determine the effectiveness of this method, the authors compare the use of Laplacian and Gaussian noise of different magnitudes. This shows that the magnitude of the noise has the most influence on attack mitigation. Secondly, the authors propose *gradient compression and sparsification*. Using this method, gradients with small magnitudes are set to zero, leading to sparsity. Upon evaluation, the authors find that the minimum level of sparsity is about 20%. The authors reason that since other works show that models can still achieve good performance when 99% of the gradients is compressed, this is a good method to avoid information leakage from gradients. Unfortunately, they do not evaluate the methods themselves. Finally, the authors suggest *changing model parameters*, such as batch size and image resolution, to reduce leakage from gradients and also evaluate its effectiveness. Here, it is important to note that these changes are model-specific. The model considered in this work is a CNN trained for image classification. Therefore, we can not be sure if this mitigation also applies to other model architectures and for other problems, such as credit card fraud detection.

3.3.3. Dropout

When training neural networks, dropout is used to prevent these from overfitting by randomly turning off certain neurons [3]. In other words, some parts of the feature vector are deleted. This is done using a *dropout rate*, representing the probability for a neuron to be turned off during training. Since activating less neurons results in less gradients available to the attacker, previous work reasons that this reduces attack success [68]. With this, it is important to note that dropout can only be used in

neural networks.

Melis et al. propose four different techniques to mitigate vulnerability to active and passive property and membership inference, among which dropout [68]. The first proposed method is to *share fewer gradients*. This means that during updates participants only share a fraction of their gradients, thereby reducing the amount of information available to the attacker. Unfortunately, Melis et al. show that even when sharing only 10% percent of the gradients, an attack still achieves an AUC score of 0.84. A similar conclusion is drawn by Phong et al. in [77]. Secondly, the authors propose using *dimensionality reduction*, either by using only samples that occur in the training data often or by restricting the model into using a pre-defined vocabulary. Their results show that the former does not work, while the latter does. However, using a pre-defined vocabulary also significantly decreases model performance. Thirdly, Melis et al. propose *dropout* as a mitigation technique. Since this results in fewer gradient updates are visible to the adversary, they reason this reduces attack success. When setting different dropout probabilities, results show a contradicting phenomenon: increasing dropout probability also increases the success of membership inference. Setting a dropout probability of 0.1, yields an AUC score of 0.94, while dropout probabilities of 0.7 and 0.9 yield an AUC score of 0.99. As a last mitigation technique, Melis et al. discuss the use of *differential privacy*. Their work demonstrates the trade-off between privacy protection and model performance: to ensure participant-level differential privacy and allow sufficient model performance at the same time, thousands of participants are required. This observation is also made in [103] and [97].

In [88], the use of *dropout* is suggested to mitigate membership inference attacks. When evaluating its use, it becomes visible that for almost all tasks dropout decreases attack performance. However, results also show that as the dropout probability increases, model performance degrades. Moreover, attacks are mitigated best for high dropout rates, which also have worst influence on model performance. Because of this, the authors conclude that a moderate dropout rate should be set. However, when inspecting attack success at this rate, it becomes visible that an adversary still reaches an attack accuracy of almost 80%, with a recall and precision of 60%.

3.3.4. Regularization

Similar to dropout, regularization is used in neural networks to prevent a model from overfitting [39]. Regularization is done by adding an extra term that is responsible for penalizing large weights to a model's cost function. This ensures that weights remain small, thereby keeping model complexity low and preventing it from overfitting.

The first work we discuss that proposes the use of regularization, also suggests several other mitigation techniques [103]. These are divided into two categories: *model hardening techniques* and *API hardening techniques*. The former category refers to methods that can be implemented during training, while the latter category refers to techniques useful during prediction. Since in this work we focus on collaborative training, we only discuss the suggested model hardening techniques. These include *model choice*, e.g. choosing a model that is proven to be resistant to inference attacks, *fit control*, e.g. preventing the model from overfitting, *regularization* by adding noise to a model's loss function and *anonymization* techniques such as k-anonymity. Unfortunately, these suggestions are not evaluated for success. Finally, the use of *differential privacy* is discussed. The authors argue that in order to determine whether differential privacy can prevent inference attacks, the formal relation between differential privacy, membership privacy, and membership inference attacks should be investigated further.

In [97], several methods to defend against membership inference are discussed. The first three measures consider *adapting the prediction vector*. As discussed earlier, such strategies are not applicable to our work. The last method suggested is to use *regularization*. With this suggestion, the authors note that regularization should be used carefully since it could negatively influence a model's performance on the test set. Unfortunately, these measures are not evaluated for success.

3.4. Multi-Party Shuffling Protocols

So far, we have discussed the vulnerability of collaborative learning to inference attacks and the proposed mitigation techniques as suggested in literature. From this, we conclude that preventing inference attacks in collaborative remains an open problem and that no suitable mitigation solution has been found so far. In this work, we reason that removing linkability between data owners and their respective model updates could prevent inference attacks since this prevents attackers from training accurate shadow models targeted at one participant specifically. In [82], a similar observation is made. Here, the authors reason that randomizing the order of training could prevent inference attacks since this prevents insider attackers from training accurate shadow models targeted at one participant. While they reason this prevents inference attacks, its evaluation is left for future work.

To remove linkability between a gradient and the participant sending it, multi-party shuffling protocols could be of use. These protocols are often designed to shuffle a given input sequence anonymously. When reviewing existing work, we found two ways to achieve this. The first method uses *mixnets*, while the second methods uses *verifiable shuffling*. These protocols are often applied in an online gambling setting to shuffle a deck of cards. To determine if protocols exist that can be readily applied to our problem, we now discuss works proposing random shuffling protocols.

The first work proposes a multi-party computation protocol that allows participants to jointly chose a permutation among themselves, while also ensuring they only know their own position in it [101]. To do so, it first creates a *re-encryption mixnet*, in which each participant forms one node of the mixnet. Then, to determine the chosen permutation, each participant is required to re-encrypt and shuffle a list of n received ciphertexts. A re-encryption algorithm is one that takes as input a ciphertext c and the public key pk it was encrypted under. As output, the algorithm gives a second ciphertext c' encrypted under the same public key. In this work, re-encryption is done using the identity message $m = 1$, thereby allowing participants to recognize their ciphertext from the complete list of ciphertexts. The protocol proceeds as follows. First, all players generate two private keys and one public key, which is shared with participant p_1. Then, participant p_1 uses these received keys to form a list of ciphertexts of length n, encrypting the identity message under each of the received public keys. Starting from p_1, each player, in turn, re-encrypts each ciphertext and shuffles the list by applying a random permutation. After participant n has completed the shuffling and re-encryption step, the list, now representing an encrypted version of the chosen permutation, is broadcast to all players. To retrieve their own position in the permutation, each participant tries to decrypt all messages in the list. Due to the re-encryption mechanism, each participant can decrypt only one ciphertext successfully, thereby informing participants of their position. To show the security of the protocol in a semi-honest setting, the authors show that the protocol has the following properties: *correctness*, ensuring that participants output a unique position in the chosen permutation, *privacy*, ensuring that no coalition of $n-2$ participants is able to learn the complete permutation, and *uniformity*, ensuring that the positions of honest players in the final permutation are distributed uniformly. Additionally, measures to make the protocol secure in a malicious setting are discussed. Similar to this work, the protocol allows participants to jointly determine a permutation among themselves. However, contrary to this work, participants know their own position in the permutation, instead of knowing the participant next in order. Additionally, the number of encryption and decryption operations performed per participant is rather high: each participant performs n re-encryption operations and n decryption operations.

In [15], a protocol is proposed that allows shuffling of a deck of cards in online poker without requiring a trusted third party. To achieve this, all players are required to participate in shuffling: each of them generates a random permutation of the card deck and keeps it secret. Additionally, each player is required to generate a commitment on the used permutation. Then, the deck is shuffled using the composition of all permutations. To ensure the deck is shuffled correctly and kept secret,

an additive homomorphic cryptosystem is used. After using the deck to play a card game, to verify that the game was played honestly, each player is required to reveal its encryption keys and the applied permutation at the end. For online poker, this is a suitable verification method since when a game has ended, the encryption keys and permutations are no longer of value. However, in our situation, we cannot reveal the permutation after training has ended: this would allow an attacker to save all received updates until the end and then perform the attack after seeing the permutation, still breaching privacy. Therefore, this protocol is not applicable to our situation.

In [112], the authors propose to verify shuffles using checksums instead of using zero-knowledge proofs. This is done by wrapping the deck with checksum information before shuffling it. Then, to prove a shuffling was done correctly, a player reveals the wrap after shuffling. In this work, two wraps are presented: one requiring a linear operation and one requiring a double exponentiation, which are added one after the other. By comparing the complexity of the protocol to related works, the authors show that their checksum-based shuffling methods have faster runtime, while also reducing the number of messages being sent. However, the results also show that when using a 2048 bit encryption key, it takes one minute to shuffle a deck. In a mental poker game, this suffices since a deck has to be shuffled only once at the beginning of the game. However, for our application, we feel less overhead is desired and should be possible.

In [73], an interactive zero-knowledge proof is proposed to verifiably shuffle a sequence of k modular integers. The output of the shuffle operation is a shuffled sequence of the same length, for which the order of elements in the output is kept secret. While the proposed interactive zero-knowledge proof is very useful in an electronic voting setting, where one group of authorities is denoted specifically for verifying the correctness of the shuffle without knowing it, in our case, we prefer to ensure a correct permutation is determined without the use of interactive zero-knowledge proofs.

In [71], two multi-party computation protocols to allow participants to determine a random permutation of their inputs while keeping the permutation itself secret are proposed. The first one is a so-called *single-output* protocol, which means that each participant receives only one of the shuffled elements. The second protocol is a so-called *all-output* multi-party shuffle, which means that all parties receive the entire shuffled sequence of inputs. The work considers a malicious adversary that is allowed to corrupt at most $t < n$ parties, where t is a security parameter. To show the security of the protocols, the authors consider the notion of *almost t-security*. This means that the outcome of the protocol satisfies the following properties: *output delivery*, ensuring that corrupted parties are unable to prevent honest parties from receiving output, and *unlinkability*, meaning that the adversary can guess the permutation correctly with a probability of at most $\frac{1}{(n-t)!}(1+\delta)$, where δ is a deviation factor. The protocol proceeds as follows. First, parties are divided in groups of $\log N$ parties, so-called *quorums*. Then, each party is assigned a quorum that it is required to share its secret input x_i with. Next, each quorum is responsible for generating a value r_i on behalf of the submitting parties. Finally, the parties in the quorum jointly sort these random numbers and share the result with a quorum one layer above. Using several layers of quorums ensures the final result contains a correct ordering of all random numbers and corresponding inputs. This corresponds to the jointly determined permutation. To evaluate the performance of the protocol, a simulation in which the number of parties is varied between 2^5 and 2^{30} is used. Results show that this work indeed improves both communication, computation, and latency cost when compared to other works. However, due to the constraint that the number of parties in quorums is defined as $\log N$, the number of participants has to be of power two. In our case, the number of participants is in the order of then. This would restrict us into either using 2^3 or 2^4 participants. We conclude that this constraint is too limiting for this protocol to be of use. Moreover, while the single-output protocol does not provide enough information, since this only informs parties on their own position in the permutation, the all-output version of the protocol provides too much information since in our case parties are not

allowed to know the complete permutation.

In [6], a multi-party computation protocol is proposed to perform secret shuffling in settings with a central server. The example setting taken into account is one where multiple companies want to compare their key performance indicators (KPIs) on benchmarks, thereby allowing them to identify how well they are doing compared to their competitors and their benchmark group. Typically, these benchmark comparisons are performed by central trusted third parties. The goal of the proposed protocol is to remove the link between the submitted data, encrypted KPIs, and the data owners, the companies. This operation is called a secret shuffle and is defined as follows [6]:

Definition 5 *Given a sequence of ciphertexts* $X = (E(x_1), ..., E(x_n)) for 1 \leq i \geq n$. *A secret shuffle* $S(\cdot)$ *is a function that, for input* X, *yields a sequence* $X = E'(x_{\pi(1)}), ..., E'(x_{\pi(n)})$ *such that* $(E(x_1) \neq E'(x_{\pi(1)}), ..., E(x_n) \neq E'(x_{\pi(n)})$ *encrypt pairwise equal plaintexts* $x_1, ..., x_n$. *The order of the elements in* X *is randomly permuted via a permutation* π. *No participant can learn more than negligibly much information about* π.

The protocol itself runs in two rounds. During the first round, each participant submits it secret input along with a random number, which both are encrypted. During the second round, the submitted values are shuffled using an interactive protocol. First, the service provider P_s applies two permutations, one to the list of encrypted input values, and one to the list of encrypted random numbers. This ensures that there no longer is a link between which participant submitted which values. Additionally, the submitted input ciphertexts are encrypted and additively blinded using random numbers to protect them from other participants. Then, the P_s separately encrypts these ciphertexts and shares the resulting list, together with the permuted inputs and encrypted random values, with the participants. Additionally, it sends a random number that participants are required to use during hashing. Upon receiving the three lists, participants decrypt the one containing the random numbers and concatenate each number with the received random number. The result is hashed and sorted. After hashing and sorting, each participant determines the index of the random number it submitted during the first step. This index is used to select a ciphertext from the encrypted list of input. This has to be re-randomized before sending it back to P_s to prevent it from recognizing the ciphertexts it receives. Additionally, a random number is added to the resulting ciphertext. Together with the ciphertext, an encrypted version of the random number is sent to the P_s. Then, during the final step, P_s decrypts all received ciphertexts, multiplies these with -1, and encrypts the result. This list is then multiplied with the other two lists to ensure the random values cancel out. The result is a random ordering of the initially submitted secret values by participants. Since each participant knows its secret value, it now knows which position it has in the permutation. This means the algorithm successfully performs a secret shuffle. However, as becomes clear from the protocol explanation, a lot of encryption and decryption operations are required to ensure this can be done. In addition, the protocol also requires a lot of messages to be sent: during multiple steps of the protocol, a quadratic number of messages is sent by the service provider.

3.5. Discussion

In this chapter, we discussed the use of collaborative learning and its parallel and collaborative settings. While this allows institutions to bundle their knowledge without sharing data, it also exposes them to powerful property and membership inference attacks. Next, we discussed the privacy-preserving mechanisms used to prevent these attacks, such as differential privacy and homomorphic encryption, dropout, and regularization. The reviewed works show that the use of homomorphic encryption results in significant computational overhead. Regarding participant-level and record-level differential privacy, it becomes clear that the number of records and participants has to be in the order of thousands to prevent degrading of model performance. Moreover, the applicability of dropout and regularization is very limited, since these techniques can only be used in

neural networks. Finally, we reviewed multi-party shuffling protocols, often used in online poker to shuffle a card deck. We reason these are useful in our situation since they enable us to break the link between data submitted by participants, such as gradients in sequential collaborative learning, and the data owners, e.g. the participants in collaborative learning. While the reviewed work indeed shows that random shuffling can be used to provide unlinkability, a protocol that meets our requirements is not yet found.

When taking all these findings into consideration in the context of collaborative credit card fraud detection, we conclude the following:

- A sequential collaborative learning setting is preferred over a parallel one since this allows us to remove the central server for aggregation.
- To mitigate the risk of inference attacks, secure multi-party shuffling protocols can be used to break the link between data and data owners.

In the remainder of this thesis, we will discuss our approach to jointly selecting a training order in a collaborative fraud detection setting.

4

Masking Private Values using Joint Random Number Generation

In this work we propose a protocol allowing parties to jointly determine an order for training without revealing it. To achieve this, first, parties are divided into two groups: *leaders* and *participants*. Then, the protocol requires participants to select an index from the permutation table randomly and to share this with the group of leaders. The group of leaders sums these selected indices using the modulus of the sum, as given in Chapter 2, to determine the final outcome of the protocol. To protect the values of the index as submitted by each participant from leaders, we mask them with random numbers. To ensure that leaders are still able to retrieve the correct modulus of the sum, we use a joint random number generation protocol that ensures the joint sum of the random numbers is zero. In other words, as the leaders sum all received indices, these numbers cancel each other out, thereby not influencing the outcome of the modulus of the sum.

In this chapter, we include a paper proposing two protocols that both allow the generation of random numbers with zero-sum. The first protocol relies on bit-wise sharing of individually generated random numbers. By requiring participants to share only one bit per round with a changing lead entity, this protocol minimizes the amount of information leaked to an adversary. The second protocol requires a single broadcast and relies on the security of the Diffie-Hellman key exchange. This protocol requires only one round to establish jointly generated numbers with a zero-sum. In this work, we use the single broadcast-based protocol to jointly generate random numbers with a zero-sum.

This chapter has been published as *"Efficient Joint Random Number Generation for Secure Multi-Party Computation"* by E. Hoogerwerf, D. van Tetering, A. Bay, and Z. Erkin in *SECRYPT.* (2021), which is presented in **Section 4.1**

Efficient Joint Random Number Generation for Secure Multi-Party Computation

Large availability of smart devices and an increased number of online activities result in extensive personalized or customized services in many domains. However, the data these services mostly rely on are highly privacy-sensitive, as in pace-makers. In the last decades, many privacy breaches have increased privacy awareness, leading to stricter regulations on data processing. To comply with this legislation, proper privacy preservation mechanisms are required. One of the technological solutions, which is also provably secure, is Secure Multi-Party Computation (SMPC) that can compute any function with secret inputs. Mainly, in several SMPC solutions, such as data aggregation, we observe that secret values distributed among parties are masked with random numbers, encrypted and combined to yield the desired outcome. To ensure correct decryption of the final result, it is required that these numbers sum to a publicly known value, for instance, zero. Despite its importance, many of the corresponding works omit how to obtain such random numbers jointly or suggest procedures with high computational and communication overhead. This paper proposes two novel protocols for Joint Random Number Generation with very low computational and communication overhead. Our protocols are stand-alone and not embedded in others, and can therefore be used in data aggregation and other applications, for instance, machine learning algorithms, that require such random numbers. We first propose a protocol that relies on bit-wise sharing of individually generated random numbers, allowing parties to adapt random numbers to yield a public sum. Second, we propose a protocol that uses the sign function to generate a random number from broadcast numbers. We provide security and complexity analyses of our protocols.

4.1. Introduction

Recent developments in machine learning and data analytics have allowed more and more customized personalized online services, thanks to the availability of vast numbers and types of data collected all around us. On the one hand, the data collected can be used to improve machine learning algorithms and provide users with personalized services such as news notifications or product recommendations. On the other hand, data collection raises serious privacy concerns since often sensitive data is included [48, 117]. Over the last years, we have witnessed numerous privacy breaches [81], damaging individuals and also businesses. These privacy breaches have led to increased privacy awareness, not only among users but also in the political arena, resulting in stricter laws on data processing such as the General Data Protection Regulation in Europe and the Personal Data Protection Act in Singapore [33, 98]. To comply with legislation, proper privacy preservation mechanisms are required. Among many other technological solutions, Secure Multi-Party Computation (SMPC) techniques attract more attention from academia and business as it provides provable security in different adversarial models.

Firstly introduced by Yao, SMPC protocols are designed to evaluate a function using secret inputs distributed among different parties, thereby yielding a public result [116]. Secure Multi-Party Computation is used to perform, among others, data clustering [32], homomorphic encryption [60] and data aggregation [31]. In this paper, we focus on a particular computation, namely Joint Random Number Generation (JRNG), which is notably used a lot in privacy-preserving data aggregation protocols. In smart meter settings, for example, individual measurements are sent to a data aggregator, responsible for computing aggregate statistics such as sum or average. Previous research has shown that these measurements, when obtained by adversaries in an attack on data confidentiality, can be used to infer patterns revealing customer behaviour [108]. To prevent such eavesdropping attacks from happening, smart meters are required to encrypt their measurements before sharing these with the aggregator. However, encryption is not sufficient to prevent malicious aggregators from inferring additional knowledge [54]. One way to achieve this and protect against malicious

aggregators is by requiring all meters to mask their measurements with random numbers. Jointly generating these among all participants in the protocol ensures they sum to a publicly known value, thereby guaranteeing the final result's correct decryption. Therefore, Joint Random Number Generation can be considered an important step in privacy-preserving data aggregation protocols.

Remarkably, most research on privacy-preserving data aggregation omits how to obtain such numbers and assumes their existence. For example, Shi et al. assume that all keys during the setup of the private stream aggregation algorithm sum to zero [94, 95]. Lu et al. assume the same in their homomorphic encryption protocol [59]. Moreover, several studies that do focus on JRNG have high computational and communication overhead. For example, Erkin and Tsudik assume a fully connected network is available, allowing each party to exchange random values with all others [31]. With large networks, this approach becomes infeasible. A similar approach by Kursawe et al. uses the notion of *leaders*, who are required to compute their random values such that all values together sum to zero [54]. This reduces the complexity for all other nodes in the network since these only communicate with the set of leaders. However, the complexity of leader nodes remains the same as in [31].

This paper presents two novel protocols to perform Joint Random Number Generation with minimal computational and communication overhead. The first protocol relies on bit-wise sharing of individually generated random numbers. By doing so, for each round, the information leaked to an adversary is minimized to only one bit per participant. The second protocol uses a single broadcast and is based on Diffie-Hellman key exchange [69]. We also present a version of the broadcast-based protocol that has reduced complexity. Finally, to compare all protocols, we conduct an analysis considering both communication and computation complexity.

Our contributions are two-fold. First, by formalizing two JRNG protocols explicitly, we provide clarity on how to efficiently perform Joint Random Number Generation. In contrast to previous work, our protocols are stand-alone and not embedded in others. Therefore, they can be used in any domain or application where JRNG is required. Finally, we show how to perform JRNG with minimal computational and communication overhead, thus improving the way JRNG is done for the whole research community.

The remainder of this paper is structured as follows. First, we discuss related work and preliminaries in Section 4.1.1 and Section 4.1.2. Next, we discuss the share-based protocol in Section 4.1.3, before discussing the broadcast-based protocol in Section 4.1.4, along with its security proof. Finally, discuss complexity in Section 4.1.5 and conclude the paper in Section 4.1.6.

4.1.1. Related work

Related work on Joint Random Number Generation (JRNG) can be divided into two categories: research that focuses on JRNG as a main goal, as does this one, and research that uses it to achieve a broader goal, and therefore embeds it in its protocol. First, we discuss work that embeds JRNG to, for example, perform privacy-preserving data aggregation: jointly generated random numbers are used to mask individual values before sending them to the aggregator. When the aggregator sums or multiplies all values, the masks cancel out and allow the aggregator to obtain the exact result without revealing individual values. Since in this work, we care about JRNG specifically, we only highlight those procedures and omit any other details. An example of such research is the work by Erkin and Tsudik, performing additive aggregation of plaintexts in a setting with at least three parties ($K \geq 3$) [31]. First, parties exchange random numbers in a pair-wise manner, i.e. parties p_i and p_j exchange values $r_{(p_i \to p_j)}$ and $r_{(p_j \to p_i)}$. Then, each party computes its random value R_{p_i} by summing over the $K-1$ received values:

$$R_{p_i} = \sum_{j=1, j \neq i}^{K} r_{(p_i \to p_j)} - r_{(p_j \to p_i)} \,. \tag{4.1}$$

It is trivial to see that the addition of these elements yields zero. While presented in a smart meter setting, this scheme can be used in any setting where JRNG is required. However, it should be noted that this work requires the presence of secure communication channels. The second work is an interactive protocol proposed by Kursawe, Danezis, and Kohlweiss [54]. Similar to Erkin and Tsudik, it jointly generates random numbers yielding a zero-sum. However, to reduce communication complexity, it uses the notion of *leaders*. At the start of the protocol, each party generates a random number and shares it with the set of leaders P. Then, each leader generates its random number in such a way that summing it with all received shares yields zero. Similar to Erkin and Tsudik, this protocol requires at least three parties and secure communication channels. The use of leaders reduces communication complexity for non-leader parties, the complexity for leaders is unchanged. This protocol is adapted by Van de Kamp et al. to remove the notion of leaders: instead, parties share their random number with all other parties [106]. In the same work, Kursawe et al. propose a JRNG scheme based on the Diffie-Hellman key exchange. The scheme yields a publicly known product rather than a sum, using the sign function $s_i(j)$.

$$s_i(j) = \begin{cases} 1 & \text{if } i > j\,, \\ -1 & \text{if } i < j\,. \end{cases} \tag{4.2}$$

In this protocol, each party broadcasts its public key $g^{k_{p_i}}$, for which $g \in \mathbb{Z}_p^*$ is a public generator and $k_{p_i} \in \mathbb{Z}_p^*$ is a random secret key. Each party uses these numbers to compute its random number R_{p_i} as follows:

$$R_{p_i} = \prod_{j=1, j\neq i}^{K} (g^{k_{p_j}})^{s_i(j)k_{p_i}} = g^{\sum_{j=1,j\neq i}^{K} s_i(j)k_{p_i}k_{p_j}}\,. \tag{4.3}$$

Verifying that this yields one is done by

$$\begin{aligned} \prod_{i=1}^{K} R_{p_i} &= g^{\sum_{i=1}^{K}\sum_{j=1,j\neq i}^{K} s_i(j)k_{p_i}k_{p_j}} \\ &= g^{\sum_{i=1}^{K}\sum_{j=1}^{i-1}(s_i(j)+s_j(i))k_{p_i}k_{p_j}} \\ &= g^0 \\ &= 1\,. \end{aligned}$$

When using this protocol for aggregation, each party masks its individual value $c_{i,j}$ using randomness R_{p_i} and sends this to the aggregator as $g^{c_{i,j}+R_{p_i}}$. As verified above, all random numbers cancel out in the final sum, leaving the aggregator with $g^{\sum_i c_{i,j}}$. Since it is not trivial to find the correct final sum from this, Kursawe et al. argue that the aggregator should launch a brute-force attack to obtain it [54]. More recently, Bell et al. embedded JRNG in an aggregation protocol by requiring parties to establish shared keys with their neighbours to derive pairwise random masks, masking input c_i as follows [7]:

$$c_i - \sum_{j<i} m_{i,j} + \sum_{j>i} m_{i,j}\,. \tag{4.4}$$

This method again ensures that all masks cancel out in the final sum. The protocol operates in a semi-honest security setting. It can be extended to a so-called semi-malicious security setting, meaning that the aggregation server is required to perform the first step of the protocol honestly, but is allowed to deviate from the protocol after that [7].

Finally, we close this section by discussing work that specifically focuses on Joint Random Number Generation. In this work, JRNG is denoted by its authors as modulo zero-sum randomness [45]. As the name explains, modulo zero-sum randomness means that when summing individual numbers, the result is zero. To ensure zero-sum randomness, three conditions have to be met:

1. *Modulo zero condition* - the sum of all random numbers X_i has to be zero.
2. *Independence condition* - all numbers $X_1, ..., X_m$ used have to be generated independently.
3. *Secrecy condition* - each player i has knowledge of the random number it generated itself, X_i, and of the fact, the sum of all numbers will yield zero, but has no additional knowledge.

In this work, to generate random numbers, the authors impose a cyclic ordering on all parties. First, each party p_i agrees on a secret random number with its two neighbors, Z_i and Z_{i-1}. Then, parties determine their final random by subtracting these two values:

$$R_{p_i} = Z_i - Z_{i-1} \,. \tag{4.5}$$

It is trivial to see that in summation, this yields zero. In follow-up work, the use of multiple access channels and verifiable quantum protocols to generate modulo zero-sum randomness is discussed [44, 46].

It is important to note that all related work in this section focuses on yielding a public sum of zero. Our approach is a more generalized one, where instead we facilitate the joint generation of random numbers that sum up to a publicly known value $N \geq 0$.

4.1.2. Preliminaries

In this section, we introduce preliminaries, the security assumption and notation used in this work.

Decisional Diffie-Hellman Problem

The security of the broadcast-based scheme presented in Section 4.1.4 relies on the Decisional Diffie-Hellman Problem (DDH). This problem is defined as follows: given a cyclic group G of prime order q, its public generator g, the following two distinguishers are computationally indistinguishable (in the security parameter q); (g^x, g^y, g^{xy}), where x, y are chosen uniformly at random from $[0, \dots q-1]$ and (g^x, g^y, g^z), where x, y, z are chosen uniformly at random from $[0, \dots q-1]$.

Security Setting

We assume an environment composed of a static group of K entities wanting to jointly establish random numbers with a publicly known sum. Similar to related work, we assume all entities to be semi-honest (honest-but-curious). This means they follow the protocol but are not prevented from inferring additional knowledge from received information. Each entity is capable of performing modular operations, such as modular addition, and of generating strong random numbers. We do not assume the presence of any other active entity, such as a trusted third party or an aggregator, to perform computations. Protecting against active adversaries can be done at the cost of additional communication and computation complexity, but is considered out of the scope of this work.

For the bit-wise sharing protocol, we assume the existence of secure communication channels. These ensure that any message sent over the network can only be retrieved by its sender and recipient.

Notation

The notation used is summarized in Table 4.1.

4.1.3. Share-based JRNG Protocol

In this section, we describe a Joint Random Number Generation protocol that uses the bit-wise sharing of individually generated random numbers. The random numbers generated by the protocol sum up to a publicly known integer $0 \leq N \leq 2^m$, for some $m \in \mathbb{N}$.

Initially, all entities generate a random number R_{p_i}, and a subset of all entities is selected as m key entities. Then, the acting key entity adjusts the j'th bit of its random number such that it

Table 4.1: Notation Summary.

Symbol	Definition
p	large prime
$r_{p_i \to p_j}$	random value generated by p_i send to p_j
g	generator for $\mathbb{Z}_p^*$
K	number of entities in the protocol
l	security parameter
m	bit-length of random numbers
C_i	carry-over from bit i
N	publicly known target value
R_{p_i}	random number generated by entity p_i
k_{p_i}	temporary values used to construct R_{p_i}
x_{ij}	bit j of random number R_{p_i}
x_{Nj}	bit j of publicly known target value N
y_i	updated bit i replacing x_{ii} of R_i
d_i	scaled carry-over for key entity i
$s_i(j)$	sign function
Z_a	short for $\mathbb{Z}/a\mathbb{Z}$

equals the j'th bit of the target N. To do so, each entity sends its j'th bit to the acting key entity, which sums these to determine if its j'th bit should change. Finally, to prevent carry-over from affecting subsequent computations, the acting key entity subtracts this from its random number. This process is repeated for m iterations until each key entity has adjusted its random number once.

The protocol is formalized as follows:

1. All K entities generate random numbers $R_{p_i} \in Z_{2^m}$, such that $R_{p_i} = \sum_{j=1}^{m} x_{ij} 2^{j-1}, x_{ij} \in \{0,1\}$. m entities are assigned key entity as in [54].

2. Then, for m iterations:

   (a) All entities i send x_{ij} to the acting key entity using the underlying secure network.

   (b) The acting key entity calculates $d_j = \sum_{i=1, i\neq j}^{K} x_{ij}$. If $d_j \equiv x_{Nj} \mod 2$, it sets its j'th coefficient to $y_j = 0$. Otherwise, it sets its j'th coefficient to $y_j = 1$.

   (c) Finally, the acting key entity calculates $C_j = \lfloor \frac{d_j + y_j}{2} \rfloor 2^j$ and updates its random number to $R_j \leftarrow R_j - C_j \mod 2^m$.

In practice, usually $N = 0$. However, to keep the final outcome secret, different values for N can be used. Then, the final outcome can only be determined by the entities that know N, since these can subtract it from the final outcome. While in this work the protocol is demonstrated in a binary setting, it can be used in any base-k setting for some $k \in \mathbb{N}$.

A distinguishing feature of this protocol is that instead of communicating full-sized random numbers in each round, this protocol uses bit-wise communication. This reduces communication complexity to a total of $K-1$ bits per iteration. This reduction becomes more significant for large numbers and high amounts of entities. Since only key entities are required to perform modular operations, the scheme can be used in situations where not all entities have equal or sufficient computing power. Even more important, the use of key entities ensures that computation complexity is dominated by the number of key entities, not by the number of entities in general. Compared to the work by Erkin and Tsudik, [31], communication complexity is reduced and a fully connected network is not required.

It is important to note that this scheme should not be used for applications where certain bits are more relevant than others, as is the case in a bank transfer. If this protocol would be used to mask a bank transfer with encryption $c = m + R \mod 2^m$, this would allow the adversary to distinguish between high-valued and low-valued transactions by only corrupting one entity.

Finally, it should be noted that the scheme can also be used for situations where $m > K$. This can be done as follows: for each j, where $K < j \leq m$, we require entity K to act during all remaining iterations, thereby continuously adjusting its random number.

Security

When assessing the share-based protocol in a semi-honest security setting, it becomes clear that the protocol is vulnerable to collusion among key entities. In each round, the acting key entity receives an entity's j'th bit, thereby obtaining partial information about each random number R_{p_i}. Because of this, collusion allows key entities to partially reconstruct the random number held by other entities. While this seemingly renders the protocol unusable, situations exist in which computational speed is more important and where non-privacy-preserving Joint Random Number Generation suffices. Such a situation is found in the work by Garay, Schoenmakers, and Villegas [37]. Here, a random number of r is jointly generated, before being encrypted using fresh randomness. By adding fresh randomness to r, it is effectively masked. Therefore, in this case, our bit-wise sharing protocol provides sufficient security.

In situations where privacy of individual bits is required, secure multi-party computation techniques, such as (additive) secret sharing can be added to the protocol [106]. This can be achieved as follows: during the second step of the protocol, instead of directly sending its bit x_{ij} to the acting key entity, each entity first generates $K-1$ shares s_{ij} of its bit, such that $x_{ij} = \sum_{j=1}^{K-1} s_{ij} \mod N$, similar to the work by Garcia and Jacobs [38]. Of these, it holds one by itself and distributes the others among other non-acting key entities. Finally, instead of sending entire bits, each entity sends the received shares to the acting key entity, together with its own share. Because of this, the acting key entity receives all shares of all bits at the same time. This allows it to reconstruct the original bits, without knowing which bit originally belonged to which entity, thereby preserving the privacy of each bit. While this allows the privacy protection of individual bits, it also leads to an increase in communication complexity. Therefore, using a privacy-preserving version of this protocol should be justified.

4.1.4. Single Broadcast-based JRNG protocol

In this section, we present a protocol, which is inspired by [54], that does not require a secure network and only needs one broadcast per entity to jointly establish random numbers with a publicly known sum of N. We again assume a group of at least three entities. Additionally, we assume a publicly known ordering exists. Similar to the protocol in [54], this protocol is based on Diffie-Hellman key exchange and uses the sign function [54]. However, different from the protocol by Kursawe et al. our protocol performs additive aggregation rather than multiplicative aggregation. Thus, we provide the additive variant here.

Before we formalize the protocol, it is important to note that in this work we assume a fully connected network. This is, however, not required since all messages can be publicly known. Therefore, different network topologies, such as a star network where all messages are relayed through a semi-honest entity, can also be used. Additionally, in this work, we set $N = 0$, while in practice any $N \in \mathbb{N}$ can be used.

1. A public generator $g \in \mathbb{Z}_p^*$ is agreed upon. All entities generate a random $k_{p_i} \in \mathbb{Z}_p^*$ and $g^{k_{p_i}}$.

2. All entities publicly broadcast $g^{k_{p_i}}$.

3. Each entity p_i computes its final random number:

$$R_{p_i} = \sum_{j=1, j\neq i}^{K} s_i(j)(g^{k_{p_j}})^{k_{p_i}} = \sum_{j=1, j\neq i}^{K} s_i(j) g^{k_{p_i} k_{p_j}}.$$

Correctness is easily verified since $s_i(j) + s_j(i) = 0$ for all $i \neq j$, and thus

$$\begin{aligned}\sum_{i=1}^{K} R_{p_i} &= \sum_{i=1}^{K} \sum_{j=1, j\neq i}^{K} s_i(j) g^{k_{p_i} k_{p_j}} \\ &= \sum_{i=1}^{K} \sum_{j=1}^{i-1} (s_i(j) + s_j(i)) g^{k_{p_i} k_{p_j}} = 0\,.\end{aligned}$$

Security

The security of this protocol is based on the hardness of Decisional Diffie-Hellman Assumption. As the protocol is a direct application of the Diffie-Hellman Key exchange protocol, its security can be reduced to Decisional Diffie-Hellman problem as R_{p_i}'s are constructed from the public-key of Diffie-Hellman Exchange protocol and can not be distinguished from random R_{p_i}'s.

Adapted Broadcast-based Protocol

To reduce the complexity of the broadcast-based scheme, a ring-topology can be assumed. This requires the assumption that any entity p_i is only able to communicate with its neighbours: the entities ranked above and below it. Since this rank is seen in $\pmod K$, the lowest and highest ranked entities are also neighbors. The scheme is formalized as follows:

1. A public generator $g \in \mathbb{Z}_p^*$ is agreed upon. All entities generate a random $k_{p_i} \in \mathbb{Z}_p^*$ and $g^{k_{p_i}}$.

2. Each entity publicly sends $g^{k_{p_i}}$ to both neighbors.

3. Each entity p_i computes

$$R_{p_i} = \sum_{j\in N_i} s_i(j)(g^{k_{p_j}})^{k_{p_i}} = \sum_{j\in N_i} s_i(j) g^{k_{p_i} k_{p_j}}.$$

Correctness follows since $i \in N_j$ implies that $j \in N_i$. Therefore, correctness of this scheme can be verified in the same way as the original scheme is verified.

4.1.5. Complexity Analysis

In this section, we analyze the complexity of our protocols. For computation complexity, we base our analysis on the number of modular additions and modular exponentiation performed. For communication complexity, we base our analysis on the number of messages exchanged and their lengths. We conclude this section by comparing our protocols.

Computation Complexity

The share-based protocol is asymmetric since the number of operations performed by key entities differs from the number of operations performed by non-key entities. This underlines the protocol's versatility: non-key entities require significantly less computing power than key entities. Its computation complexity is dominated by modular addition, performed each round by the acting key entity using all $K-1$ received values. For m iterations, this results in $m \cdot (K-1)$ modular additions. It is important to note that in this protocol modular addition is performed using single bits. In contrast to the share-based protocol, the single broadcast-based protocol is symmetric, meaning that the number of operations performed is the same for all entities. Besides modular addition, the

Table 4.2: Computation and communication complexity of existing and our works.

	Modular Addition	Modular Exp.	Number of Messages	Message length (bits)	Encryption/ Decryption	Key Agreement
[31]	$K \cdot (K-1)$	-	K^2	m	-	-
[54] DH key exchange based	$K \cdot (K-1)$	$2K$	K^2	m	-	-
[54] Interactive	$P \cdot (K-1)$	-	KP	m	KP/KP	-
[106]	$K \cdot (K-1)$	-	K^2	m	-	-
[7]	$K \cdot (K-1)$	-	$2K$	m	-	$2K$
[45]	K	-	$2K$	m	-	$2K$
Our Share-based pro.	$m \cdot (K-1)$	-	K	1	-	-
Our Broadcast-based pro.	$K \cdot (K-1)$	K	K^2	m	-	-
Our Adapted broadcast-based pro.	$2K$	K	$2K$	m	-	-

complexity of the broadcast-based protocol is dominated by modular exponentiation. Each entity performs both operations once during the protocol: modular exponentiation during the first step and modular addition during the final step. This results in K modular exponentiations and $K \cdot (K-1)$ modular additions, since modular addition is performed using $(K-1)$ numbers. In contrast to the first protocol, the addition is performed using complete numbers. Compared to the original version of the broadcast-based protocol, its adapted version has lower computation complexity. This is caused by the fact that entities only send messages to their two neighbors. As a consequence, the amount of numbers used in modular addition is reduced to 2 and computation complexity to $2K$. The number of modular exponentiation remains the same.

Communication Complexity

For the share-based JRNG protocol, communication complexity strongly depends on the number of iterations m performed during one run of the protocol. For each iteration, each entity sends one message to the acting key entity. Consequently, for m iterations, this results in m messages send by non-key entities and $m-1$ messages send by key entities. For K entities, this yields a communication complexity of $O(mK)$. This is comparable to any other system using the notion of l leaders, as discussed in Related Work. However, since our protocol requires entities to share one bit of their random number only, communication complexity is reduced, even if m far exceeds l. The single broadcast-based protocol uses broadcasting to communicate. Therefore, its communication complexity is rather high. Each entity sends one message to all others, resulting in $K \cdot (K-1)$ messages being sent during one execution of the protocol, simplified as K^2. Additionally, compared to the share-based protocol, messages are longer, since numbers are communicated entirely. In the adapted version of the single broadcast-based protocol, each entity only sends two messages, thereby reducing communication complexity to $2K$. Similar to the original version of the protocol, numbers are communicated entirely.

Comparison

Computation and communication complexity are shortly summarized in Table 4.2. When looking at communication complexity, it becomes visible that both the share-based and the adapted protocol require a number of messages that is linear with K. The share-based protocol requires the shortest messages and therefore has the lowest communication complexity. When taking into account computation complexity, it becomes visible that, again, the share-based and adapted protocol have the lowest complexity. Since the share-based protocol does not require modular exponentiation, while the others do, it has the best computation complexity. However, it is important to note that it does require the existence of secure communication channels, while messages in the broadcast-based protocol can be shared publicly

In Table 4.2, we also include computation and communication complexity of related works. Comparing these to the share-based protocol shows that the share-based protocol has better computation and communication complexity, as it requires less modular additions and sends fewer and shorter messages. Similar to [31], it requires secure communication channels. Table 4.2 also shows the computation and communication complexity of the broadcast-based protocol are similar to those of the works by [31, 54] and [106]. Additionally, while the works by [7] and [45] have better communication complexity, they require additional key agreements to be performed. The same holds for the interactive protocol by [54], while computation and communication complexity are similar, it requires additional encryption and decryption operations.

4.1.6. Conclusion

While Joint Random Number Generation can be found in several Secure Multi-Party Computation solutions, such as data aggregation, most research omits how to obtain such numbers and simply assumes their existence. In this paper, we have presented two computationally efficient protocols for Joint Random Number Generation. One of these is share-based, using bit-wise updates to ensure that all individual numbers yield the correct sum. While this protocol does not provide semi-honest security, situations exist in which the current security guarantees suffice and where computational speed is more important. For completeness, we also demonstrate measures to increase the security of the share-based protocol. The other protocol is broadcast-based and uses the sign function to ensure that all generated numbers yield a public sum. This protocol relies on the computational Diffie-Hellman Problem to provide semi-honest security. Additionally, we have proposed an adapted version of the broadcast-based protocol that has lower computation and communication complexity. By comparing all three protocols based on computation and communication complexity, it becomes clear that the share-based protocol is most efficient. However, the fact that it requires secure communication channels hinders its applicability. Therefore, when choosing which protocol to employ, the desired security guarantees and the additional cost of establishing secure channels should be taken into account.

Acknowledgement

This work was partly supported by SECREDAS project that has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 783119. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Netherlands, Austria, Belgium, Czech Republic, Germany, Spain, Finland, France, Hungary, Italy, Poland, Portugal, Romania, Sweden, Tunisia, United Kingdom.

5

Joint Permutation Selection for Sequential Collaborative Learning

In this work, we consider a setting with multiple parties that want to collaboratively train a predetermined machine learning model in a sequential setting. With this protocol, we prevent inference attacks by jointly choosing the training order without informing all parties about the result completely. To achieve this, we divide parties into two groups: *leaders* and *participants*. At the start of the protocol, leaders agree on a permutation table π containing all $n!$ permutations of participants, representing all possible orders for training. To determine the selected permutation, each participant is required to submit a random number $0 \leq v_i \leq (n! - 1)$. Since sharing the permutation with each participant directly still enables the execution of inference attacks, participants are only informed on which participant is next in order and should receive their data. This is done using majority voting: each leader informs each participant on who to send their data by voting on the respective participant. Participants send their data to the one that has received the most votes. This ensures the resulting order is correct as long as the majority of the leaders are honest.

Our protocol broadly works as follows:

- *Initialization:* A set of leaders $A_1, \ldots, A_\ell$ is selected randomly. Together, these agree on the permutation table π. Additionally, public parameters for the encryption scheme are set. Based on these, all parties generate their public and secret key. Finally, leaders jointly generate the shared decryption key.

- *Joint Random Number Generation:* All participants jointly generate random numbers with a zero-sum. These are used later to mask individual submissions.

- *Joint Index Selection:* Each participant generates a random number $0 \leq v_i \leq (n! - 1)$ and encrypts it together with the jointly generated random number from the previous step. The resulting ciphertext is shared with the leaders.

- *Aggregation & Partial Decryption:* Each leader uses additive homomorphism to sum the received ciphertexts. Additionally, it shares a partial decryption of the result with all other leaders. Finally, each leader combines all received partial decryptions to determine the selected index v using the modulus of the sum.

- *Majority Voting:* To inform each participant who to send their data to, leaders send each participant a ballot with a vote for the respective participant. Each participant tallies the received votes and sends its data to the participant that received the most votes.

Adversarial Setting

We assume a fixed group of parties that are willing to collaboratively train a model of a predetermined type. Each party p_i owns a local dataset and has published its public key pk_{p_i}. We assume each party has sufficient computational power to train a model locally in a reasonable time. Additionally, we assume that parties agreed on features and data structure in advance.

All parties are divided into two groups: a set of ℓ leaders $A_1, \ldots, A_\ell$ and a group of n participants $p_1, \ldots, p_n$, for which $\ell \leq n$. From the protocol description, it is evident that leaders can assert significant influence over the outcome and correct execution of the protocol. Since leaders know the final permutation that is chosen, they could for example manipulate the order of training by sending participants false information. Therefore, we assume leaders to be *malicious* and require them to provide non-interactive zero-knowledge proofs to prove their correct behavior.

In contrast to leaders, participants have far less influence on the outcome of the protocol: during joint index selection, each participant is asked to supply a random number. This means that as long as one participant does so honestly, the chosen permutation is determined randomly. Therefore, we assume participants to be *semi-honest* (honest-but-curious). This means they follow the protocol honestly but are not prevented from inferring additional knowledge using information received during the execution of the protocol.

Notation

Table 5.1: Notation Summary.

Symbol	Definition
p	large prime
g	generator for $\mathbb{Z}_p^*$
q	(prime) order of g
G	cyclic group with prime order q
ℓ	number of leaders
$A_1, \ldots, A_\ell$	group of leaders
n	number of participants
$p_1, \ldots, p_n$	group of participants
$n!$	number of possible permutations of participants
v_i	random number generated by p_i, $0 \leq v_i \leq (n! - 1)$
v	jointly selected permutation index
π	permutation table containing all permutations
π_v	jointly selected permutation v from π
β	lookup table for Shanks' Baby Step Giant Step algorithm
sk_{p_i} / sk_{A_i}	secret key held by p_i / A_i
pk_{p_i} / pk_{A_i}	public key held by p_i / A_i
PK	shared decryption key held by leaders
k_{p_i}	secret value generated by p_i
R_{p_i}	jointly generated random number of p_i
r_i	fresh randomness in $\mathbb{Z}_q^*$
$E(c_{1_{p_i}}, c_{2_{p_i}})$	ciphertext created by p_i
$H(c_1^{sk_{A_i}})$	commitment on c_1 by A_i
$s_i(j)$	sign function
$ZKP(c_{pk_{pi}}(c_i))$	non-interactive zero-knowledge proof on ciphertext c_i encrypted using pk_{p_i}

Protocol 5.1: Initialization

All leaders $A_1, \ldots, A_\ell$ do the following:

- Generate permutation table π
- Setup distributed exponential ElGamal (G, p, q, g)
- Generate lookup table β
- Generate public and secret key pair (sk_{A_i}, pk_{A_i})
- Generate shared decryption key $PK = \prod_{i=1}^{\ell} pk_{A_i} = g^{x_1 + \cdots + x_\ell}$

All participants $p_1, \ldots, p_n$ do the following:

- Generate public and secret key pair (sk_{p_i}, pk_{p_i})

5.1. Initialization

During initialization, the set of leaders $A_1, \ldots, A_\ell$ is selected deterministically, similar to [53]. Together, the leaders agree on a permutation table π, containing all possible permutations $n!$. Next, leaders agree on all public parameters of the protocol such as the parameters of the distributed exponential ElGamal cryptosystem: a large prime p, a cyclic group G of prime order q, and a public generator g. Using these, they compute the lookup table β, used to compute the discrete log of g^v during the aggregation & partial decryption protocol. Next, all participants generate their secret key sk_{p_i} and public key pk_{p_i} using the key generation algorithm described in Chapter 2. Leaders generate their secret key sk_{A_i} and public key pk_{A_i} similarly. Finally, leaders generate the shared decryption key using Equation 2.12.

An overview of the initialization procedure is given in Protocol 5.1.

5.1.1. Generating the Permutation Table

The permutation table is used by leaders $A_1, \ldots, A_\ell$ to determine the final result of the protocol. In other words, each row of the table represents one possible ordering of participants $p_1, \ldots, p_n$. To ensure all leaders retrieve the same result at the end of the protocol, it is important that this table is agreed upon in advance. This can be done in several ways. For example, leader A_1 could generate a first version of the table, after which other leaders are allowed to submit permutations that are applied one after the other to create the final table. An even simpler way of generating the table is to task one leader with generating it and requiring this leader to share the table with all others. Since the table is allowed to be publicly known, its correctness can be easily verified by any leader or participant.

An example of a permutation table π with three participants p_1, p_2 and p_3 is given below:

Table 5.2: Example of permutation table π with three participants p_1, p_2 and p_3.

Index v	Permutation π_v
1	p_1, p_2, p_3
2	p_1, p_3, p_2
3	p_2, p_1, p_3
4	p_2, p_3, p_1
5	p_3, p_2, p_1
6	p_3, p_1, p_2

Protocol 5.2: Joint Random Number Generation

All participants $p_1, \ldots, p_n$ do the following:

- Generate secret value k_{p_i} and broadcast $g^{k_{p_i}}$ to all participants
- Compute final random number using $R_{p_i} = \sum_{j=1, j\neq i}^{K} s_i(j) g^{k_{p_j}} \cdot g^{k_{p_i}}$

5.1.2. Shanks' Baby Step Giant Step Algorithm

During the aggregation & partial decryption phase of the protocol, we require leaders to determine the chosen permutation v from g^v using the lookup table β. This is a consequence of using the exponential ElGamal cryptosystem. To determine v, the discrete log of g^v has to be computed. In other words, we have to solve $h = g^v \pmod p$ for v, given a prime p. We do so by using Shanks' baby step giant step algorithm [91]. This is done by first defining a value $m = \lceil \sqrt{(p-1)} \rceil$ and rewriting v as $v = i \cdot m + j$, for some $0 \leq i, j < m - 1$. From this, we can derive the following equations:

$$h = g^v \pmod p = g^{im+j} \pmod p \,, \tag{5.1}$$

$$g^j = h \cdot g^{-im} \pmod p \,. \tag{5.2}$$

Consequently, finding values for i and j such that Equation 5.2 holds, results in finding the correct value for v. We do so by first performing the baby step and then performing the giant step. During the baby step, a lookup table β containing values of $h \cdot g^{-im}$ for $0 \leq i < m - 1$ is computed. During the giant step, the lookup table is used to perform trial multiplication by computing g^j.

To speed up the aggregation & partial decryption phase, we pre-compute the lookup table during initialization. Doing so allows us to compute the table only once, whereas generating it during aggregation & partial decryption requires us to generate it ℓ times. Therefore, in the end, generating it during initialization is most efficient in regards to runtime of the protocol.

5.2. Joint Random Number Generation

Joint random number generation (JRNG) is performed among participants. Together, they derive random numbers that sum to zero. These numbers are used in the next phase, joint index selection, to mask individual values while also ensuring their sum can still be derived. To achieve this, we use the single broadcast-based joint random number generation protocol as discussed in Chapter 4. This protocol uses the sign function $s_i(j)$:

$$s_i(j) = \begin{cases} 1 & \text{if } i > j \,, \\ -1 & \text{if } i < j \,. \end{cases} \tag{5.3}$$

In short, the protocol is executed as follows. First, each participant generates a secret value $k_{p_i} \in \mathbb{Z}_p^*$ and uses this to compute $g^{k_{p_i}}$, using the public generator g of the exponential ElGamal cryptosystem. Next, each participant broadcasts $g^{k_{p_i}}$ to all other participants. Then, each participant computes its final random number using the values received from other participants as follows:

$$R_{p_i} = \sum_{j=1, j\neq i}^{K} s_i(j) g^{k_{p_j}} \cdot g^{k_{p_i}} \,. \tag{5.4}$$

An overview of the JRNG procedure is given in Protocol 5.2.

Protocol 5.3: Joint Index Selection

All participants $p_1, \ldots, p_n$ do the following:

- Generate random $0 \leq v_i \leq (n!-1)$ and fresh randomness $r_i \in \mathbb{Z}_q^*$
- Encrypt $E(c1_{p_i}, c2_{p_i}) = (g^{r_i}, g^{v_i} \cdot PK^{r_i} \cdot g^{R_{p_i}})$
- Broadcast $E(c1_{p_i}, c2_{p_i})$

5.3. Joint Index Selection

During joint index selection, each participant first selects a permutation from the permutation table individually. Then, all selections are shared with the leaders to determine the final selection. To achieve this, each participant generates a random number $0 \leq v_i \leq (n!-1)$ indicating the individually selected permutation. Then, this value is masked using the random number R_{p_i} generated during joint random number generation. The masked result is encrypted using the distributed exponential ElGamal cryptosystem and fresh randomness $r_i \in \mathbb{Z}_q^*$. Altogether, this yields $E(c1_{p_i}, c2_{p_i})$:

$$\begin{aligned} c1_{p_i} &= g^{r_i}\,, \\ c2_{p_i} &= g^{v_i} \cdot PK^{r_i} \cdot g^{R_{p_i}}\,. \end{aligned} \tag{5.5}$$

Each participant shares $E(m_{p_i}) = E(c1_{p_i}, c2_{p_i})$ with all leaders.

An overview of the Joint Index Selection procedure is given in Protocol 5.3.

5.4. Aggregation & Partial Decryption

After joint index selection, each leader aggregates all received ciphertexts using additive homomorphism to obtain its ciphertext value. During aggregation, the masks computed using joint random number generation cancel out, allowing correct addition of ciphertexts. This is verified as follows:

$$\begin{aligned} E(m_{p_1}) \cdot \cdots \cdot E(m_{p_n}) &= (g^{r_1}, g^{v_1} \cdot PK^{r_1} \cdot g^{R_{p_1}}) \cdot \cdots \cdot (g^{r_n}, g^{v_n} \cdot PK^{r_n} \cdot g^{R_{p_n}}) \\ &= (g^{r_1+\cdots+r_n}, g^{v_1+\cdots+v_n} \cdot PK^{r_1+\cdots+r_n} \cdot g^{R_{p_1}+\cdots+R_{p_n}}) \\ &= (g^{r_1+\cdots+r_n}, g^{v_1+\cdots+v_n} \cdot PK^{r_1+\cdots+r_n} \cdot g^{0}) \\ &= (g^{r_1+\cdots+r_n}, g^{v_1+\cdots+v_n} \cdot PK^{r_1+\cdots+r_n} \cdot 1) \\ &= E(m_{p_1} + \cdots + m_{p_n})\,. \end{aligned}$$

After obtaining the aggregated result, to determine the plaintext value of the selected index, a partial decryption is published by each leader, together with a commitment $H(c_1^{sk_{A_i}})$. If all commitments are verified, decryption proceeds as follows:

$$\frac{c_2}{\prod_{i=1}^{\ell} c_1^{sk_{A_i}}} = \frac{c_2}{c_1^{sk_{A_1}+\cdots+sk_{A_n}}} = g^{v}\,. \tag{5.6}$$

Finally, v is determined using lookup table β.

An overview of this procedure is given in Protocol 5.4.

Hash Commitment Schemes

In this work, we use hash commitment schemes based on the cryptographic hash function SHA-256. As discussed in Chapter 2, a commitment scheme has to satisfy two properties. First of all, it has to be *hiding*, ensuring a Verifier is unable to retrieve a value x from its commitment $c(x)$. This is guaranteed by the fact that cryptographic hash functions are *one-way* [80]:

Protocol 5.4: Aggregation & Partial Decryption

All leaders $A_1, \ldots, A_\ell$ do the following:

- Aggregate all received ciphertexts $\prod_{i=1}^{n} E(m_{p_i})$
- Broadcast partial decryption and commitment $(c_1^{sk_{A_i}}, H(c_1^{sk_{A_i}}))$
- Computed complete decryption $\frac{c_2}{\prod_{i=1}^{\ell} c_1^{sk_{A_i}}} = g^v$
- Determine v using a lookup table

Definition 6 *A hash function is one-way in the sense that given a Y in the image of h, it is hard to find a message X such that $h(X) = Y$, and given X and $h(X)$ it is hard to find a message $X' \neq X$ such that $h(X') = h(X)$.*

Secondly, a commitment has to be *binding*, ensuring a Prover cannot reveal a different $x' \neq x$, for which $c(x) = c(x')$. This is ensured by the *collision resistance* of cryptographic hash functions [80]:

Definition 7 *A collision resistant hash function is a one-way hash function h for which it is hard to find two distinct messages that hash to the same result.*

So, in order to create hash commitments that satisfy the hiding and binding properties, a one-way, collision resistant hash function has to be used. Since SHA-256 satisfies both these properties, we can use SHA-256 to construct our commitments.

Determining v using Shanks' Baby Step Giant Step Algorithm

To determine v from $g^v \pmod p$ we use Shanks' baby step giant step algorithm [91]. As the baby step was already performed during initialization, during aggregation & partial decryption the giant step is performed. This means that a value $0 \leq j < m$ has to be found that satisfies:

$$g^j = h \cdot g^{-im} \pmod p \tag{5.7}$$

This is done using trial multiplication: trying values for j until we find one that satisfies the relation defined above. This value represents the value for v that was used to obtain $g^v \pmod p$.

5.5. Majority Voting

In this work, we use majority voting to detect malicious leaders that might inform participants on who to send their data incorrectly. The use of majority voting allows us to detect these leaders using aggregation of received votes. As long as the majority of leaders follow the protocol correctly, the result of the vote is correct and participants are informed on who to send their data to correctly.

Majority voting proceeds as follows: using the obtained index v, resulting from aggregation & partial decryption, leaders determine the corresponding permutation π_v by selecting the v'th row from π. From this, leaders determine to which participant p_j a participant p_i should send its data. To inform participants about this, leaders create n punch-hole vector ballots B_{p_i}, as described in Chapter 2, together with the corresponding non-interactive zero-knowledge proofs.

After receiving ℓ ballots and verifying zero-knowledge proofs, participants perform homomorphic tallying of votes by forming matrices using received ballots. This means each row represents a received ballot B_{p_i} and each cell (B_{p_i}, p_j) indicates whether a leader A_i voted to send data to participant p_j or not. By summing each column using additive homomorphism, the number of leaders agreeing on a participant p_j is determined, informing p_i who to send its data to.

An overview of the majority voting step is given in Protocol 5.5.

Protocol 5.5: Majority Voting

All leaders $A_1, \ldots, A_\ell$ do the following:

- Determine π_v
- Create n ballots $B_{p_1}, \ldots, B_{p_n}$ and send these to the corresponding participant

All participants $p_1, \ldots, p_n$ do the following:

- Aggregate all received ballots B_{p_i} using additive homomorphism
- Send data to participant p_j with the most votes

5.5.1. Determining the Selected Permutation

The selected permutation is determined using the permutation table π and the permutation index v obtained during partial decryption & aggregation. For example, when using the permutation table π as shown in Table 5.3 and for $v = 2$, we get the following permutation: $\pi_2 = p_1, p_3, p_2$.

Table 5.3: Example of permutation table π with three participants p_1, p_2 and p_3.

Index v	Permutation π_v
1	p_1, p_2, p_3
2	p_1, p_3, p_2
3	p_2, p_1, p_3
4	p_2, p_3, p_1
5	p_3, p_2, p_1
6	p_3, p_1, p_2

5.5.2. Generating Ballots

Each leader informs each participant who to send their data to using punch-hole vector ballots and non-interactive zero-knowledge proofs (ZKP). For the same example permutation as selected above, $\pi_2 = p_1, p_3, p_2$, this means that participant one, p_1, is required to send its data to participant three, p_3, who is required to send its data to participant two, p_2. Finally, p_2 is required to send its data to p_1. For each leader A_i that follows the protocol honestly, this results in several ballots B_{p_i}, shown in Table 5.4.

Table 5.4: Example of ballots B_{p_i} each leader A_i generates for permutation p_1, p_3, p_2.

B_{p_i}	p_1	p_2	p_3
B_{p_1}	-	$c_{pk_{p1}}(0), ZKP(c_{pk_{p1}}(0))$	$c_{pk_{p1}}(1), ZKP(c_{pk_{p1}}(1))$
B_{p_2}	$c_{pk_{p2}}(1), ZKP(c_{pk_{p2}}(1))$	-	$c_{pk_{p2}}(0), ZKP(c_{pk_{p2}}(0))$
B_{p_3}	$c_{pk_{p3}}(0), ZKP(c_{pk_{p3}}(0))$	$c_{pk_{p3}}(1), ZKP(c_{pk_{p3}}(1))$	-

5.5.3. Tallying Ballots

As discussed in Chapter 2, using the Exponential ElGamal cryptosystem allows homomorphic additive tallying of votes. This is done by participants to learn who to send their data to: after tallying the received votes, a participant will send its data to the participant that received the most votes. In Table 5.5, we include an example tallying result from one participant, who has received four votes of which three leaders, A_1, A_2, and A_4 are honest and one leader, A_3, is dishonest for the chosen

permutation $\pi_2 = p_1, p_3, p_2$. This table also shows how tallying of the votes allows us to identify the misbehaving leader: since three votes are cast for sending data to p_3 and only one vote was cast for sending data to p_2, A_3 is identified as a misbehaving leader.

Table 5.5: Example ballots B_{p_1} generated by leaders $A_1, \ldots, A_4$ for participant p_1.

A_i	p_1	p_2	p_3
A_1	-	$c_{pk_{p1}}(0), ZKP(c_{pk_{p1}}(0))$	$c_{pk_{p1}}(1), ZKP(c_{pk_{p1}}(1))$
A_2	-	$c_{pk_{p1}}(0), ZKP(c_{pk_{p1}}(0))$	$c_{pk_{p1}}(1), ZKP(c_{pk_{p1}}(1))$
A_3	-	$c_{pk_{p1}}(1), ZKP(c_{pk_{p1}}(1))$	$c_{pk_{p1}}(0), ZKP(c_{pk_{p1}}(0))$
A_4	-	$c_{pk_{p1}}(0), ZKP(c_{pk_{p1}}(0))$	$c_{pk_{p1}}(1), ZKP(c_{pk_{p1}}(1))$
Tallied result	-	1	3

6

Analyzing the Joint Permutation Selection Protocol

In this chapter, we perform a security, complexity and performance analysis of our joint permutation selection protocol as presented in Chapter 5.

6.1. Security Analysis

In this section, we provide a sketch of security of our protocol, which relies on the security of the underlying procedures. Therefore, we will discuss security of each of the underlying procedures and their building blocks separately. To ensure a permutation is selected from the permutation table correctly, the protocol relies on the security of the joint random number generation phase and the joint index selection phase. To ensure a participant is informed on who to send their data to correctly, the protocol relies on the security of the aggregation & partial decryption and the majority voting phase.

6.1.1. Joint Random Number Generation

The security of the joint random number generation phase is based on the security of the single broadcast-based joint random number generation protocol as discussed in Chapter 4. For this protocol, the security of the final random number R_{p_i} is based on the computational Diffie-Hellman assumption, stating that no adversary is able to solve the computational Diffie-Hellman problem in polynomial time with non-negligible probability. Additionally, the security of the individually generated random numbers k_{p_i} is guaranteed by the Diffie-Hellman Key exchange [26], requiring that each value k_{p_i} is shared with other parties as $g^{k_{p_i}}$. Because of this, an attacker trying to find the secret value k_{p_i} has to solve the Discrete Logarithm Problem. It is clear that if an adversary could solve the Discrete Logarithm Problem (DLP) and derive k_{p_i} from $g^{k_{p_i}}$, it could also solve the CDH problem with one exponentiation. Therefore, DLP is at least as hard as CDH. Because of the hardness of the CDH problem and because the Diffie Hellman Key Exchange protocol is secure against semi-honest adversaries, the joint random number generation protocol is secure in a semi-honest setting. Additionally, since each final random number R_{p_i} is composed of other random numbers k_{p_i}, its value is random as long as one entity submits its value honestly.

6.1.2. Joint Index Selection

As discussed in Chapter 5, during joint index selection we require each participant to select an index $0 \leq v_i \leq (n!-1)$ randomly. Because of this, the final value v is random as long as one participant submits its random number honestly. This means that even when a majority of participants, up to $n-1$, do not submit their random value v_i honestly, the final permutation π_v is still chosen randomly.

6.1.3. Aggregation & Partial Decryption

To ensure the correct value of v is derived from the values submitted by participants, threshold decryption is used. This is done by first requiring each leader to submit a partial decryption, along with a commitment. The commitment ensures that leaders are unable to change their partial decryption value after sharing it with the other leaders. Additionally, it ensures that leaders are unable to change their value after sharing it.

6.1.4. Majority Voting

To ensure participants are informed correctly on who to send their data to, the protocol relies on the security of the majority voting protocol. To ensure each ballot is correctly formed and can not be manipulated by malicious leaders, leaders are required to include non-interactive zero-knowledge proofs to prove a ciphertext is indeed encrypted from its plaintext value. The use of majority voting ensures that as long as the majority of leaders informs participants correctly the outcome of the vote is correct.

6.1.5. Overall Security

With this analysis, we show that all steps performed by leaders are secure against malicious adversaries, while steps performed by participants are secure against semi-honest ones. Because of this, our protocol is secure against malicious leaders and semi-honest participants.

6.2. Complexity Analysis

In this section, we analyze the computation and communication complexity of our protocol. For computation complexity, we base our analysis on the number of modular additions, multiplications and exponentiations that are performed. Additionally, we take into account the number of encryption and decryption operations. For communication complexity, we take into account the number of messages send. Finally, we take into account the number of required non-interactive zero-knowledge proofs. In our analysis we distinguish between operations performed by leaders and operations performed by participants. An overview of the computation and communication complexity of our protocol per leader and per participant is given in Table 6.1.

In this section, we assess the computation complexity of each phase and distinguish between leaders and participants.

6.2.1. Initialization

During initialization, each leader performs modular exponentiation once to generate its public key. Additionally, to establish the shared public key, each leader performs ℓ modular multiplications. Participants are required to perform one modular exponentiation to generate their public key.

6.2.2. Joint Random Number Generation

The joint random number generation protocol is performed by participants only. It requires n modular exponentiations, one for each participant to compute $g^{k_{p_i}}$. Additionally, each participant sums $n-1$ received values to compute its final random number. In other words, n participants sum $n-1$ numbers, resulting in a total of $n \cdot (n-1)$ modular additions being performed by participants.

Since each participant is required to broadcast its value for $g^{k_{p_i}}$ to all other participants, each participant sends $n-1$ messages. Therefore, during joint random number generation a total of $n \cdot (n-1)$ messages is send by participants.

6.2.3. Joint Index Selection

Similar to joint random number generation, the joint index selection protocol is performed by participants only. When looking at computation complexity it becomes visible that each participant

Table 6.1: Computation and communication complexity of our protocol.

	Modular Addition	Modular Multiplication	Modular Exponentiation	Encryption/ Decryption	Number of Messages	Non-interactive ZKP
Leaders $A_1, \ldots, A_\ell$						
1. Initialization	-	ℓ	1	-	$\ell - 1$	-
2. Joint Random Number Generation	-	-	-	-	-	-
3. Joint Index Selection	-	-	-	-	-	-
4. Aggregation & Partial Decryption	-	n	1	-/1	$\ell - 1$	-
5. Majority Voting	-	-	-	$n \cdot (n-1)$/-	n	$n \cdot (n-1)$
Participants $p_1, \ldots, p_n$						
1. Initialization	-	-	1	-	-	-
2. Joint Random Number Generation	$n-1$	-	1	-	$n-1$	-
3. Joint Index Selection	-	-	-	1/-	ℓ	-
4. Aggregation & Partial Decryption	-	-	-	-	-	-
5. Majority Voting	-	$\ell \cdot (n-1)$	-	-/$\ell \cdot (n-1)$	1	-

performs encryption once. Since the resulting ciphertext is shared with each leader, this results in ℓ messages being send per participant.

6.2.4. Aggregation & Partial Decryption

The aggregation & partial decryption phase is performed by leaders only. First, to aggregate the received ciphertexts, each leader performs n modular multiplications. Then, each leader computes a partial decryption $c_1^{sk_{A_\ell}}$, which requires one modular exponentiation. Finally, each leader performs one decryption operation to obtain the value g^v. Since each leader shares its partial decryption value with all other leaders, $\ell - 1$ messages are send.

6.2.5. Majority Voting

During the majority voting phase, leaders first generate one ballot for each participant. Since each ballot consists of $n-1$ options, this results in $n \cdot (n-1)$ encryption operations performed by each leader. Since we require leaders to include a non-interactive zero-knowledge proof for each ciphertext, the number of non-interactive zero-knowledge proofs is the same.

After receiving all ballots, each participant aggregates them using additive homomorphism. This results in $n-1$ modular multiplication operations per participant. After doing so, each participant performs $n-1$ decryption operations on the resulting aggregated ciphertexts to obtain the final result of the protocol.

Overall Computation Complexity

When looking at Table 6.1, it becomes clear that for both leaders and participants computation complexity is dominated by the complexity of the majority voting phase. For leaders, the complexity of this phase is defined by the number of encryption operations and non-interactive zero-knowledge proofs required. For participants, it is determined by the number of modular multiplications, required to aggregate received votes, and the number of decryption operations, required to determine which participant received the most votes. It should be noted that for situations in which semi-honest behavior of leaders provides sufficient security guarantees this phase can be left out. When leaders are assumed to be semi-honest, they will not inform participants incorrectly, thereby

eliminating the need for majority voting.

Theoretical Bound on Computation Complexity

To determine the total computation complexity for leaders, we sum the computation complexity of all procedures. This gives the following:

$$\ell + 1 + n + 1 + 1 + n \cdot (n-1) = O(n^2) \,. \tag{6.1}$$

Since we know $\ell \leq n$, this can be simplified as $O(n^2)$ computation complexity for leaders.
Doing the same for participants gives:

$$1 + (n-1) + 1 + 1 + \ell \cdot (n-1) + \ell \cdot (n-1) = O(n^2) \,. \tag{6.2}$$

This is also simplified as $O(n^2)$. This means that for both leaders and participants, our protocol has quadratic computation complexity.

Overall Communication complexity

When taking into account communication complexity, it becomes visible that for leaders several procedures contribute to the total communication complexity somewhat equally: both initialization and aggregation & partial decryption require leaders to send $\ell - 1$ messages. Additionally, the majority voting procedure requires leaders to send n messages. For participants, communication complexity is mostly determined by the joint random number generation procedure. Participants are also required to messages during joint index selection but since we know $\ell \leq n$, we assume the number of messages send during joint random number generation is higher. Moreover, in practice often the number of leaders ℓ will be much less than the number of participants n, $\ell << n$. Therefore, in practice the number of messages send during joint random number generation will be much more compared to the number of messages send during joint index selection. From this we can conclude that in general communication complexity is higher for participants.

Theoretical Bound on Communication Complexity

To determine the total communication complexity for leaders, we sum the number of messages required during each procedure. This yields:

$$(\ell - 1) + (\ell - 1) + n = O(n) \,. \tag{6.3}$$

Since we know $\ell \leq n$, this can be simplified as $O(n)$ communication complexity for leaders.
Doing the same for participants results in:

$$(n-1) + \ell + 1 = O(n) \,. \tag{6.4}$$

Again, this can be simplified as $O(n)$. Consequently, our protocol has a linear communication complexity for both leaders and participants.

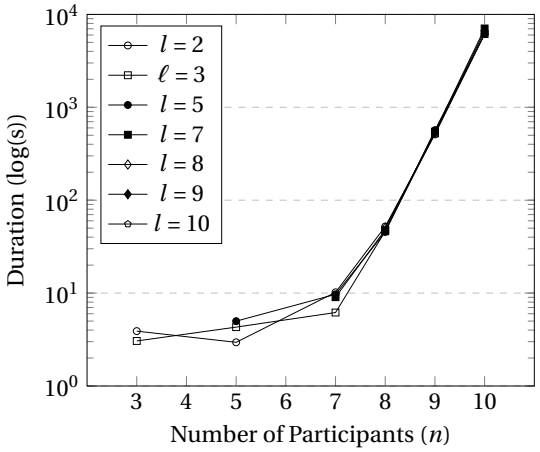
6.3. Performance Analysis

Since our protocol is the first to leverage secure multi-party computation techniques to prevent inference attacks in a sequential collaborative learning setting, we provide a practical performance analysis. This is done by implementing the protocol and measuring runtime of each of its steps, for varying numbers of participants and leaders. The number of leaders evaluated are $[2,3,5,7,8,9,10]$, while the number of participants varies between $[3,5,7,8,9]$. For each experiment, the number of leaders is less than or equal to the number of participants, $\ell \leq n$. Each setting was run ten times to prevent the influence of outliers.

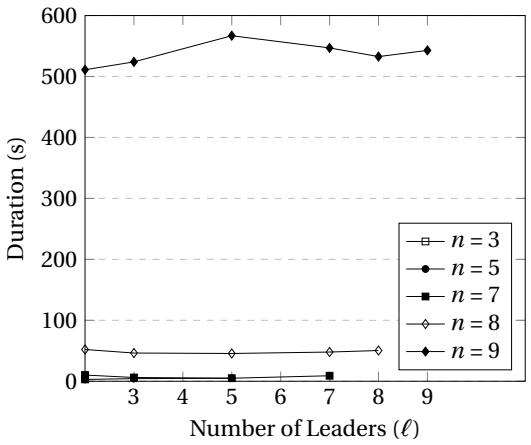
Independence of ℓ

Manual inspection of the results showed that while the number of participants n has great influence, the runtime of the protocol is independent of the number of leaders ℓ. To illustrate this, we use the initialization protocol as an example. Below, we illustrate the influence of the number of participants for varying numbers of leaders in Figure 6.1a and the influence of the number of leaders for varying numbers of participants in Figure 6.1b.

The first graph shows that when increasing the number of participants, the slopes of the lines start



(a) Average duration (log(s)) over 10 runs of initialization for varying numbers of participants (n).



(b) Average duration (log(s)) over 10 runs of initialization for varying numbers of leaders (ℓ).

Figure 6.1: Figures showing that duration of initialization is independent of the number of leaders ℓ.

to increase faster. This means that when the number of participants is increased, initialization duration increases with this. Additionally it becomes visible that all lines are very close, almost on top of each other. This shows that an increase in protocol duration is mostly caused by an increase in the number of participants. The second graph shows that increasing the number of leaders has very little influence on the duration of the protocol: the lines depicted in the graph have an almost horizontal slope. Additionally, the distance between the depicted lines gets bigger as the number of participants increases: the first three lines, for $n = 3, 5, 7$, are on top of each other, while the distance between the next two lines increases fast as the number of participants increases. Altogether, this shows that the duration of the initialization protocol is independent of the number of leaders. When doing so for the other protocols, this shows that for these protocols the same holds. Therefore, to prevent unnecessary cluttering of the figures, in the remainder of this analysis we fix the number of leaders to $\ell = 3$.

All experiments were conducted using a 2048-bit key. All tests were performed using a Dell XPS 15 9550 laptop, with an Intel Core i7 CPU and 16GB RAM. The protocol was implemented using Python 3.9 and using the gmpy2 library to perform modular arithmetic[1].

6.3.1. Initialization

Runtime of the initialization step is dominated by pre computations, such as the generation of the lookup table required for leaders to complete decryption. The length of the lookup table is defined as $n \cdot n!$, where n denotes the number of participants. As the length of the table increases with a factorial factor, this means the time needed for initialization increases similarly as the number of participants increases. Figure 6.2 indeed shows that when increasing the number of participants from $n = 7$ to $n = 8$, the average duration of the initialization protocol increases significantly. The

[1] https://gmpy2.readthedocs.io/en/latest/

same can be seen when increasing the number of participants from eight to nine.

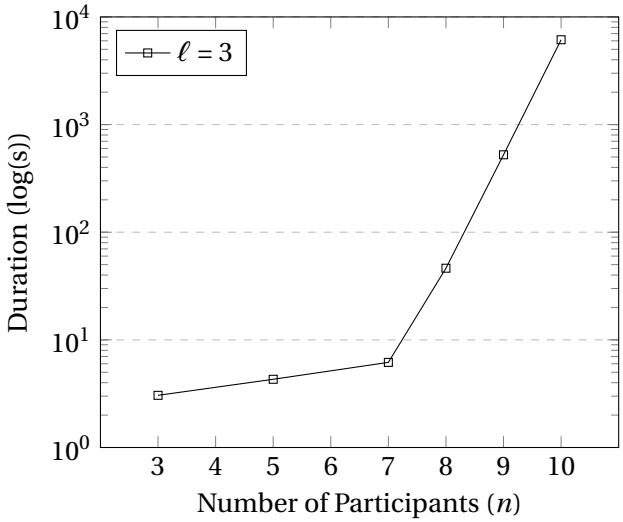


Figure 6.2: Average duration (log(s)) over 10 runs of initialization for varying numbers of participants (n).

6.3.2. Joint Random Number Generation

Joint random number generation is performed among participants only. In contrast to the initialization protocol, the number of operations does not increase with a factorial factor. Therefore, its duration is expected to increase slower. This is also visible from Figure 6.3. While the duration of joint random number generation increases when the number of participants increases, overall the protocol requires less than half a second to finish.

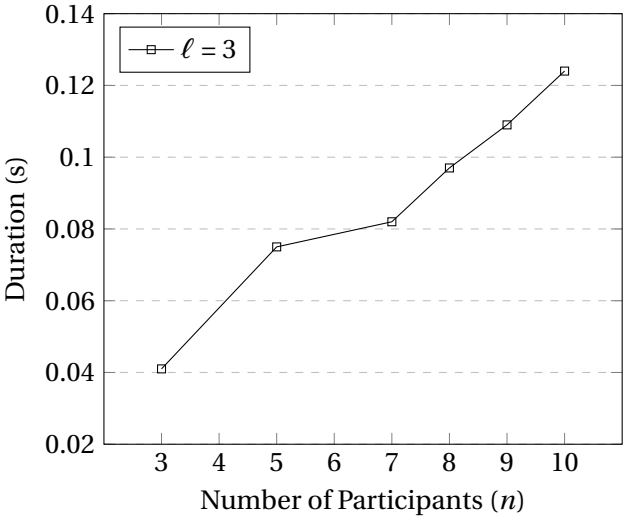


Figure 6.3: Average duration (s) over 10 runs of joint random number generation for varying numbers of participants (n).

6.3.3. Joint Index Selection

The heaviest computation in this protocol is encryption, performed by participants using their generated numbers v_i. After encryption, each participant shares its ciphertext with all leaders. The average duration of the joint index selection protocol is shown in Figure 6.4. Again, its duration increases as the number of participants increases. This is because the number of encryption opera-

tions performed increases with it. The required encryption operations also explain why this protocol requires more runtime compared to the joint random number generation protocol.

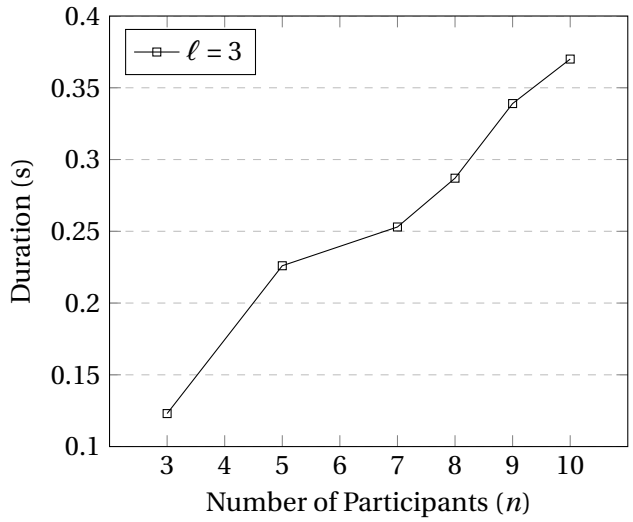


Figure 6.4: Average duration (s) over 10 runs of joint index selection for varying numbers of participants (n).

6.3.4. Aggregation & Partial Decryption

During aggregation & partial decryption, each leader first performs partial decryption and full decryption afterwards. Finally, to determine the value for v, each leader uses the lookup table of length $n \cdot n!$, as generated during the initialization protocol. From this, it follows that duration of this protocol grows significantly as the number of participants increases. This is also visible from the runtime results shown in Figure 6.5. Since in this protocol we perform a lookup rather than generating the table itself, the duration of aggregation & partial decryption is less compared to initialization.

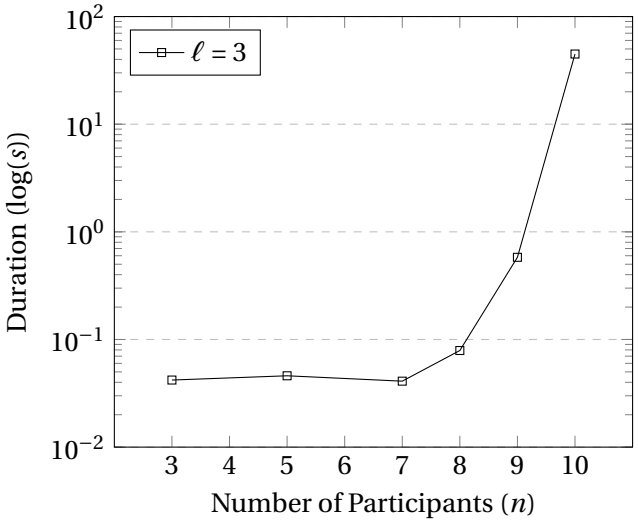


Figure 6.5: Average duration (log(s)) over 10 runs of aggregation & partial decryption for varying participants (n).

6.3.5. Majority Voting

The start of the majority voting protocol requires each leader to create n encrypted ballots, which are shared with corresponding participants. After receiving ℓ ballots, each participant sums these

and decrypts the result. From this, it becomes clear that as the number of participants increases, the number of ballots increases with it. This also results in more encryption operations being performed. Figure 6.6 shows that the duration of the majority voting protocol is similar to that of the joint index selection protocol.

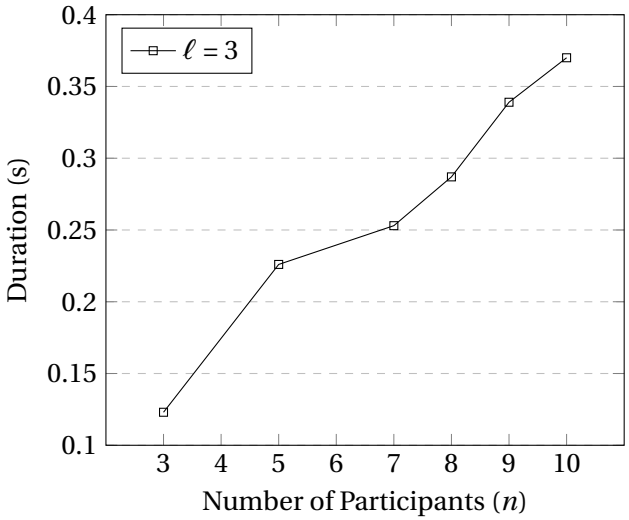


Figure 6.6: Average duration (s) over 10 runs of majority voting for varying numbers of participants (n).

Overall performance

So far, we assessed the performance of each protocol separately, for varying numbers of leaders and participants. To ensure our experiments match a collaborative learning setting realistic in industry or academia, we ensured the number of leaders was never higher than the number of participants. Additionally, while the number of leaders varied starting from two, the number of participants varied from three. This is necessary to ensure security: in a collaborative learning setting with only two participants, inferring whether a sample is owned by the other participant is obvious.

Taking into account the duration of each protocol separately shows that all protocols run in at most half a seconds, except for initialization and aggregation & partial decryption. The reason that these two protocols are significantly more slow compared to the others is that we require the use of Shanks' baby step giant step algorithm to determine the jointly selected permutation v from the permutation table π. With this, it is important to note that initialization has to be performed only once. Additionally, the duration of aggregation & partial decryption is still less than a second for all $n \leq 9$. Altogether, after initialization inference attacks are prevented during training with only a few seconds overhead.

7

Discussion and Future Work

Since millions of Euros are lost due to credit card fraud each year, Automated Fraud Detection Systems are needed. Currently, performance of these systems is obstructed by a lack of positive fraud samples in collected transaction data. To increase this number, a collaborative learning setting, in which participants jointly train a machine learning model, can be employed. In this thesis, we discussed the current state-of-the-art regarding collaborative machine learning. This showed that most work requires the existence of a central entity, responsible for aggregating locally trained models. Moreover, previous research has shown that employment of these settings is hindered by their vulnerability to inference attacks, aiming to extract secret knowledge from a model about the data it was trained on. In this work, we presented a system that allows privacy-preserving collaborative learning without the need to share data, that is also robust against inference and timing attacks. The main research question in this work was as follows:

How can we design a privacy-preserving collaborative learning system that increases the number of positive samples available during training, while introducing only minimal computation and communication overhead and providing robustness against inference attacks?

In this chapter, we discuss how our system achieves our research goal. Additionally, we discuss its limitations and how these can be improved upon in the future.

7.1. Discussion

In our attempt to answer our research question, we have taken into account two additional aspects. The first one describes how we aim to increase the number of positive samples during training without sharing data, neither directly nor in an encrypted format, while the second describes how we do not wish to employ a central entity. These aspects were addressed during our literature review. When reviewing existing work, we found two settings that allow collaborative learning without the need to share data: *parallel* collaborative learning and *sequential* collaborative learning. Therefore, by focusing on techniques suitable for these settings only, we ensured data sharing among participants was not required. Additionally, when comparing works that used either of these settings, our literature review showed that while a *parallel* setting requires a central entity to aggregate locally trained models, a *sequential* setting does not. Therefore, we decided to employ a sequential setting in our system, thereby ensuring a central entity is not required.

In our further attempt to answer our research question, we first compared privacy-preserving techniques such as homomorphic encryption and differential privacy and determined whether these could be applied in our setting. Next, we reviewed multi-party shuffling since these would enable us to break the link between data and the participants that submitted it. From this, no protocol

was found that could be readily applied in our setting. Therefore, we designed our own multi-party selection protocol that allows participants to jointly select a training order in a collaborative fraud detection setting. To provide robustness against inference attacks, we ensure that after selecting the order for training, participants are only informed on which participant should receive their data. By keeping the participant they receive data from secret, participants are unable to attack one specific participant, thereby preventing inference attacks.

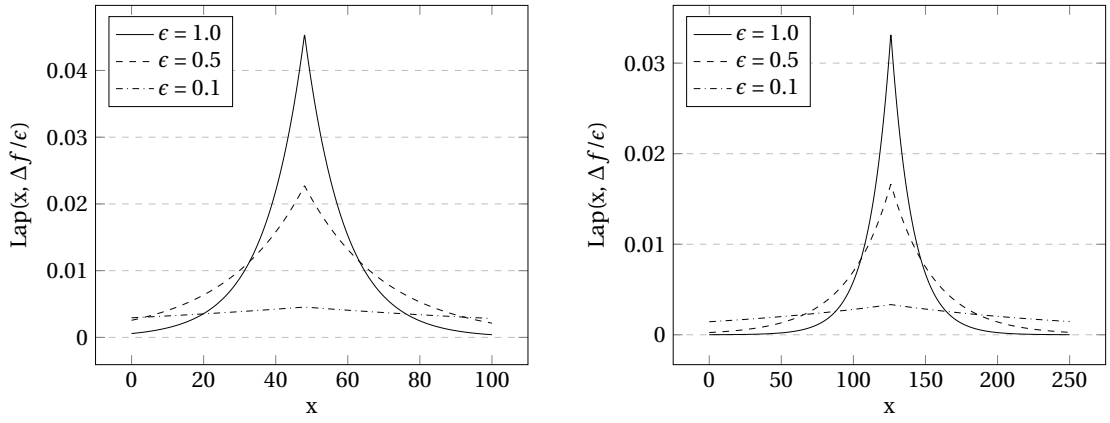
To show that we introduce minimal overhead, we take into account computation and communication complexity of our system. The analysis performed in 6 shows that for both leaders and participants the protocol has quadratic computation complexity and linear communication complexity. Since these bounds are theoretical, we conducted a runtime performance analysis to get a better insight into what contributed to the computational bound most. Results show that, even though the initialization and aggregation & partial decryption procedures take much more time compared to the other procedures, altogether, the total runtime of the protocol is less than a second for all settings with less than nine participants. With this, we show that our protocol provides minimal communication and computation overhead. Therefore, by using a multi-party selection protocol, we can increase the number of positive samples available during training, while introducing only minimal computation and communication overhead.

Vulnerability to Timing Attacks

Our protocol as presented so far has a vulnerability to timing attacks, caused by the assumption that computations by each participant take equal time. After jointly selecting the permutation, the first party starts training and sends its gradients to the next participant after doing so. This process is repeated by each participant following the determined permutation. When assuming that all participants own the same amount of data and use the same hardware for training, this results in equal training times. Using this information, participants obtain their position in the permutation by dividing the total time they have waited by the time needed for training. Now, inference attacks can be executed by attacking the first and second models received and by comparing results obtained in the attacks. For example, in a membership inference attack participant p_{n-1} can accurately attack participant p_n. This is done by first launching, for example, a membership inference attack on the model as received during the first iteration. From this, participant p_{n-1} learns that the data sample x was not held by the participants that trained the model before it. Then, during the second cycle, it performs the attack again and now it learns that the data sample x was held by the participants that trained the model before it. Since the participants knows that there was only one participant training the model after it, it can now infer that the data sample x is held by participant p_n. The same attack can be executed by participant p_2, targeted at p_1. Additionally, all other participants can determine whether a data sample was held by the before or after them, thereby also acquiring secret knowledge.

To mitigate this vulnerability, we propose to introduce random delays, which are determined using the Laplace mechanism $\text{Lap}(\Delta f/\epsilon)$ [63, 89]. By using values for ϵ that are known to provide sufficient privacy, such as 0.1, 0.5, or 1.0, we provide differentially private training times [30]. The sensitivity Δf, is determined using the average and the maximum of the training times of the last t rounds. To ensure each participant has access to the correct value Δf, we require leaders to share it with the participants. In Figure 7.1, we include several random delays as generated for different values of sensitivity Δf and privacy parameter ϵ. For the first $t-1$ rounds, we use prior knowledge of training times that participants might have, as they are likely to have worked with the data before. Alternatively, a broad Laplace distribution that becomes more narrow as new time measurements come in can be used until enough measurements are obtained to compute the average.

To show that the discussed timing attack is mitigated by using the Laplace mechanism, we show that given a time $t_{p_i} = t_{train_{p_i}} + t_{delay_{p_i}}$, an attacker is unable to distinguish between the value of



(a) Laplace distribution for $\Delta f = 11$ and $avg = 48$.

(b) Laplace distribution for $\Delta f = 15$ and $avg = 126$.

Figure 7.1: Laplace distributions for varying values of epsilon ϵ, sensitivity Δf and average avg.

$t_{train_{p_i}}$ and that of $t_{delay_{p_i}}$. To show this, we first look at the situation without delays. Here, the total time t is defined as $t = \sum_{i=i}^{n} t_{train_{p_i}}$. For equal training times, this is equal to $t = n \cdot t_{train_{p_0}}$. Then, a position in the permutation is determined using $i = \frac{n \cdot t_{train_{p_0}}}{t_{train_{p_0}}}$. When taking into account the situation including Laplacian delays $\text{Lap}(\Delta t_{train}/\epsilon)$, this yields $t = \sum_{i=i}^{n} t_{train_{p_i}} + n \cdot \text{Lap}(\Delta t_{train}/\epsilon) = n \cdot t_{train_{p_0}} + n \cdot \text{Lap}(\Delta t_{train_{p_0}}/\epsilon)$ for total time. Consequently, the position in the permutation is given by $i = \frac{n \cdot t_{train_{p_0}}}{t_{train_{p_0}} + \text{Lap}(\Delta t_{train_{p_0}}/\epsilon)}$. As the attacker does not know how much delay is added by using Laplacian noise, this prevents the participant from determining its position. Additionally, since the delay is differentially private and differs each round, a participant is unable to infer its value using information obtained over subsequent rounds. Altogether, this shows that the discussed timing attacks are mitigated using Laplacian delays.

7.2. Future Work

Even though the presented protocol shows promising results towards mitigating inference attacks in collaborative learning and thereby enabling collaborative credit card fraud detection, several limitations were identified that can be improved upon. In this section, we discuss each limitation and how to improve it separately.

Exponential ElGamal cryptosystem

Encryption in our protocol is done using exponential ElGamal to allow additively homomorphic aggregation of ciphertexts. However, under this cryptosystem messages are encrypted as g^m instead of m. Therefore, during decryption, to retrieve the original plaintext message, the discrete log has to be retrieved. We do so by generating a lookup table during the initialization procedure. Then, during the aggregation & partial decryption, we use the lookup table to determine the correct plaintext value. The size of the lookup table is given by $n \cdot n!$ for n participants. Therefore, as the number of participants increases, the duration of the protocol increases significantly with it. To improve this, a different additively homomorphic cryptosystem, such as distributed Paillier could be used. While setting up the public parameters for this cryptosystem is slightly more complicated due to the fact that the RSA modulus N has to be constructed from two large, secret primes, using this cryptosystem could allow for a speedup of the aggregation & partial decryption procedure. Therefore, using a different distributed, additively homomorphic cryptosystem could improve overall protocol performance. This, in turn, could increase the number of participants that can participate in the protocol without causing a significant increase in protocol runtime.

Evaluating the protocol in a real-world setting
Since the protocol is designed to allow collaborative credit card fraud detection in industry, for example among financial institutions in the Netherlands, it would be interesting to deploy it and to compare its detection rate to detection rates obtained when using centralized, non-collaborative models. To achieve this, an industry partner would be required. Currently, no collaboration has been established. However, it is encouraging to see that Dutch banks are starting to recognize the advantages of collaborative fraud detection. For example, Rabobank discusses the advantages of using secure multi-party computation techniques for collaborative fraud detection in its 'Technology Trend Report' of 2020 [83]. Additionally, five Dutch banks in the Netherlands have founded 'Transaction Monitoring Netherlands' (TMNL) that aim to collaboratively monitor their payment transactions to improve detection of criminal money flows and financing of terrorism [74]. This shows that in the future, collaboration with industry is possible.

Extending security to the malicious model
In this work, we have considered the setting of credit card fraud detection as an example setting that would benefit of doing so collaboratively. In this setting, we feel that the assumption of semi-honest participants and malicious leaders provides sufficient security guarantees. However, since our protocol is versatile and does not use domain-related knowledge, it can also be employed in any other setting that could benefit from sequential collaborative learning or multi-party selection. To provide sufficient security in these settings, extending security to the malicious model could be required. Therefore, extending security to the malicious increases applicability of our system.

7.3. Conclusion

The performance of automated Fraud Detection Systems is hindered by a lack of positive samples in collected transaction data. While this problem could be solved by using sequential collaborative learning, its vulnerability to powerful inference attacks restricts its applicability. In this work, we presented a protocol to prevent the execution of inference attacks using secure multi-party computation techniques. To achieve this, we require parties to jointly determine the training order. While doing so, we ensure that participants only receive information on whom to send their data to. This disables them from training a model using parameters received from a specific target participant, thereby preventing the execution of inference attacks. By including a security analysis, we have shown our protocol is robust against semi-honest participants and malicious leaders, as long as the majority of leaders behaves honestly. To prove our protocol is practical, we analysed its complexity and performance in settings realistic in industry and academia. Results show our protocol prevents inference attacks while requiring an overhead of only a few seconds. With this work, we provide privacy and robustness to inference attacks with minimal overhead. To our knowledge, our work is the first to leverage secure multi-party computation techniques to prevent the execution of inference attacks in sequential collaborative learning.

Bibliography

[1] Muhammad Asad, Ahmed Moustafa, and Chao Yu. A critical evaluation of privacy and security threats in federated learning. *Sensors*, 20(24):7182, 2020. doi: 10.3390/s20247182. URL https://doi.org/10.3390/s20247182.

[2] John O Awoyemi, Adebayo O Adetunmbi, and Samuel A Oluwadare. Credit card fraud detection using machine learning techniques: A comparative analysis. In *2017 International Conference on Computing Networking and Informatics (ICCNI)*, pages 1–9. IEEE, 2017.

[3] Pierre Baldi and Peter J. Sadowski. Understanding dropout. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2814–2822, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/71f6278d140af599e06ad9bf1ba03cb0-Abstract.html.

[4] European Central Bank. Payment statistics: 2019, 2020. URL https://www.ecb.europa.eu/press/pr/stats/paysec/html/ecb.pis2019~71119b94d1.en.html. Last Accessed: June 18, 2021.

[5] European Central Bank. Sixth report on card fraud, 2020. URL https://www.ecb.europa.eu/pub/cardfraud/html/ecb.cardfraudreport202008~521edb602b.en.html. Last Accessed: June 18, 2021.

[6] Kilian Becher and Thorsten Strufe. Efficient cloud-based secret shuffling via homomorphic encryption. In *IEEE Symposium on Computers and Communications, ISCC 2020, Rennes, France, July 7-10, 2020*, pages 1–7. IEEE, 2020. doi: 10.1109/ISCC50000.2020.9219588. URL https://doi.org/10.1109/ISCC50000.2020.9219588.

[7] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269, 2020.

[8] Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. Fast and differentially private algorithms for decentralized collaborative machine learning. *CoRR*, abs/1705.08435, 2017. URL http://arxiv.org/abs/1705.08435.

[9] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191. ACM, 2017. doi: 10.1145/3133956.3133982. URL https://doi.org/10.1145/3133956.3133982.

[10] Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011. URL http://arxiv.org/abs/1106.1813.

[11] Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognit.*, 30(7):1145–1159, 1997. doi: 10.1016/S0031-3203(96)00142-2. URL https://doi.org/10.1016/S0031-3203(96)00142-2.

[12] Stephen S Burns and Mickey W Kowitz. Voice controlled wireless communication device system, June 7 2011. US Patent 7,957,975.

[13] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 1999. doi: 10.1007/3-540-48910-X_8. URL https://doi.org/10.1007/3-540-48910-X_8.

[14] Rémi Canillas, Rania Talbi, Sara Bouchenak, Omar Hasan, Lionel Brunie, and Laurent Sarrat. Exploratory study of privacy preserving fraud detection. In *Proceedings of the 19th International Middleware Conference, Middleware Industrial Track 2018, Rennes, France, December 10-14, 2018*, pages 25–31. ACM, 2018. doi: 10.1145/3284028.3284032. URL https://doi.org/10.1145/3284028.3284032.

[15] Jordi Castellà-Roca, Josep Domingo-Ferrer, Andreu Riera, and Joan Borrell. Practical mental poker without a TTP based on homomorphic encryption. In Thomas Johansson and Subhamoy Maitra, editors, *Progress in Cryptology - INDOCRYPT 2003, 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003, Proceedings*, volume 2904 of *Lecture Notes in Computer Science*, pages 280–294. Springer, 2003. doi: 10.1007/978-3-540-24582-7_21. URL https://doi.org/10.1007/978-3-540-24582-7_21.

[16] Ken Chang, Niranjan Balachandar, Carson K. Lam, Darvin Yi, James M. Brown, Andrew Beers, Bruce R. Rosen, Daniel L. Rubin, and Jayashree Kalpathy-Cramer. Distributed deep learning networks among institutions for medical imaging. *J. Am. Medical Informatics Assoc.*, 25(8): 945–954, 2018. doi: 10.1093/jamia/ocy017. URL https://doi.org/10.1093/jamia/ocy017.

[17] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptol.*, 1(1):65–75, 1988. doi: 10.1007/BF00206326. URL https://doi.org/10.1007/BF00206326.

[18] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan Su. This looks like that: Deep learning for interpretable image recognition. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8928–8939, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/adf7ee2dcf142b0e11888e72b43fcb75-Abstract.html.

[19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 395–412. ACM, 2019. doi: 10.1145/3319535.3363207. URL https://doi.org/10.1145/3319535.3363207.

[20] Junjie Chen, Wendy Hui Wang, and Xinghua Shi. Differential privacy protection against membership inference attack on machine learning for genomic data. In *Biocomputing*

2021: Proceedings of the Pacific Symposium, Kohala Coast, Hawaii, USA, January 3-7, 2021. WorldScientific, 2021. doi: 10.1142/9789811232701_0003. URL https://doi.org/10.1142/9789811232701_0003.

[21] Trishul M. Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In Jason Flinn and Hank Levy, editors, *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, pages 571–582. USENIX Association, 2014. URL https://www.usenix.org/conference/osdi14/technical-sessions/presentation/chilimbi.

[22] Chung-Cheng Chiu, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J. Weiss, Kanishka Rao, Ekaterina Gonina, Navdeep Jaitly, Bo Li, Jan Chorowski, and Michiel Bacchiani. State-of-the-art speech recognition with sequence-to-sequence models. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*, pages 4774–4778. IEEE, 2018. doi: 10.1109/ICASSP.2018.8462105. URL https://doi.org/10.1109/ICASSP.2018.8462105.

[23] Amadi Emmanuuel Chukwudi, Eze Udoka, and Ikerionwu Charles. Game theory basics and its application in cyber security. *Advances in Wireless Communications and Networks*, 3(4): 45–49, 2017.

[24] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. Microsoft azure. *Apress: New York, NY, USA*, 2015.

[25] Ivan Damgård. Commitment schemes and zero-knowledge protocols. In Ivan Damgård, editor, *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark, July 1998*, volume 1561 of *Lecture Notes in Computer Science*, pages 63–86. Springer, 1998. doi: 10.1007/3-540-48969-X_3. URL https://doi.org/10.1007/3-540-48969-X_3.

[26] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976. doi: 10.1109/TIT.1976.1055638. URL https://doi.org/10.1109/TIT.1976.1055638.

[27] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. Technical report, 2004. URL http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html.

[28] Wei Du, Ang Li, and Qinghua Li. Privacy-preserving multiparty learning for logistic regression. In Raheem Beyah, Bing Chang, Yingjiu Li, and Sencun Zhu, editors, *Security and Privacy in Communication Networks - 14th International Conference, SecureComm 2018, Singapore, August 8-10, 2018, Proceedings, Part I*, volume 254 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 549–568. Springer, 2018. doi: 10.1007/978-3-030-01701-9_30. URL https://doi.org/10.1007/978-3-030-01701-9_30.

[29] Cynthia Dwork. Differential privacy: A survey of results. In Manindra Agrawal, Ding-Zhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation, 5th International Conference, TAMC 2008, Xi'an, China, April 25-29, 2008. Proceedings*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2008. doi: 10.1007/978-3-540-79228-4_1. URL https://doi.org/10.1007/978-3-540-79228-4_1.

[30] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014. doi: 10.1561/0400000042. URL https://doi.org/10.1561/0400000042.

[31] Zekeriya Erkin and Gene Tsudik. Private computation of spatial and temporal power consumption with smart meters. In *International Conference on Applied Cryptography and Network Security*, pages 561–577. Springer, 2012.

[32] Zekeriya Erkin, Thijs Veugen, Tomas Toft, and Reginald L Lagendijk. Privacy-preserving distributed clustering. *EURASIP Journal on Information Security*, 2013(1):4, 2013.

[33] European Parliament and Council of European Union. Regulation (eu) 2016/679, 2016. Available at: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN, Last Accessed: June 18, 2021.

[34] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *J. Cryptol.*, 1(2): 77–94, 1988. doi: 10.1007/BF02351717. URL https://doi.org/10.1007/BF02351717.

[35] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. doi: 10.1007/3-540-47721-7_12. URL https://doi.org/10.1007/3-540-47721-7_12.

[36] Arik Friedman and Assaf Schuster. Data mining with differential privacy. In Bharat Rao, Balaji Krishnapuram, Andrew Tomkins, and Qiang Yang, editors, *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 493–502. ACM, 2010. doi: 10.1145/1835804.1835868. URL https://doi.org/10.1145/1835804.1835868.

[37] Juan Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *International Workshop on Public Key Cryptography*, pages 330–342. Springer, 2007.

[38] Flavio D Garcia and Bart Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In *International Workshop on Security and Trust Management*, pages 226–238. Springer, 2010.

[39] Benyamin Ghojogh and Mark Crowley. The theory behind overfitting, cross validation, regularization, bagging, and boosting: Tutorial. *CoRR*, abs/1905.12787, 2019. URL http://arxiv.org/abs/1905.12787.

[40] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, 1999. doi: 10.1145/293411.293443. URL https://doi.org/10.1145/293411.293443.

[41] Maoguo Gong, Jialun Feng, and Yu Xie. Privacy-enhanced multi-party deep learning. *Neural Networks*, 121:484–496, 2020. doi: 10.1016/j.neunet.2019.10.001. URL https://doi.org/10.1016/j.neunet.2019.10.001.

[42] Google. Ai platform documentation, 2021. URL https://docs.aws.amazon.com/machine-learning/latest/dg/what-is-amazon-machine-learning.html.

[43] Thore Graepel, Kristin E. Lauter, and Michael Naehrig. ML confidential: Machine learning on encrypted data. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*, volume 7839 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2012. doi: 10.1007/978-3-642-37682-5_1. URL https://doi.org/10.1007/978-3-642-37682-5_1.

[44] Masahito Hayashi. Secure modulo sum via multiple access channel. *arXiv preprint arXiv:1812.10862*, 2018.

[45] Masahito Hayashi and Takeshi Koshiba. Secure modulo zero-sum randomness as cryptographic resource. *IACR Cryptol. ePrint Arch.*, 2018:802, 2018. URL https://eprint.iacr.org/2018/802.

[46] Masahito Hayashi and Takeshi Koshiba. Verifiable quantum secure modulo summation. *arXiv preprint arXiv:1910.05976*, 2019.

[47] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I. P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In Yan Chen, Alvaro A. Cárdenas, Rachel Greenstadt, and Benjamin I. P. Rubinstein, editors, *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, AISec 2011, Chicago, IL, USA, October 21, 2011*, pages 43–58. ACM, 2011. doi: 10.1145/2046684.2046692. URL https://doi.org/10.1145/2046684.2046692.

[48] Linshan Jiang, Xin Lou, Rui Tan, and Jun Zhao. Differentially private collaborative learning for the iot edge. In *EWSN*, pages 341–346, 2019.

[49] Johannes Jurgovsky, Michael Granitzer, Konstantin Ziegler, Sylvie Calabretto, Pierre-Edouard Portier, Liyun He-Guelton, and Olivier Caelen. Sequence classification for credit-card fraud detection. *Expert Syst. Appl.*, 100:234–245, 2018. doi: 10.1016/j.eswa.2018.01.037. URL https://doi.org/10.1016/j.eswa.2018.01.037.

[50] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019. URL http://arxiv.org/abs/1912.04977.

[51] Aggelos Kiayias and Moti Yung. The vector-ballot e-voting approach. In Ari Juels, editor, *Financial Cryptography, 8th International Conference, FC 2004, Key West, FL, USA, February 9-12, 2004. Revised Papers*, volume 3110 of *Lecture Notes in Computer Science*, pages 72–89. Springer, 2004. doi: 10.1007/978-3-540-27809-2_9. URL https://doi.org/10.1007/978-3-540-27809-2_9.

[52] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. Secure logistic regression based on homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2018:74, 2018. URL http://eprint.iacr.org/2018/074.

[53] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. Privacy-friendly aggregation for the smart-grid. In Simone Fischer-Hübner and Nicholas Hopper, editors, *Privacy Enhancing Technologies - 11th International Symposium, PETS 2011, Waterloo, ON, Canada, July 27-29, 2011. Proceedings*, volume 6794 of *Lecture Notes in Computer Science*, pages 175–191. Springer, 2011. doi: 10.1007/978-3-642-22263-4_10. URL https://doi.org/10.1007/978-3-642-22263-4_10.

[54] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. Privacy-friendly aggregation for the smart-grid. In Simone Fischer-Hübner and Nicholas Hopper, editors, *Privacy Enhancing Technologies - 11th International Symposium, PETS 2011, Waterloo, ON, Canada, July 27-29, 2011. Proceedings*, volume 6794 of *Lecture Notes in Computer Science*, pages 175–191. Springer, 2011. doi: 10.1007/978-3-642-22263-4_10. URL https://doi.org/10.1007/978-3-642-22263-4_10.

[55] Chester Leung. Towards privacy-preserving collaborative gradient boosted decision tree learning. 2020.

[56] Jing Li, Xiaohui Kuang, Shujie Lin, Xu Ma, and Yi Tang. Privacy preservation for machine learning training and classification based on homomorphic encryption schemes. *Inf. Sci.*, 526:166–179, 2020. doi: 10.1016/j.ins.2020.03.041. URL https://doi.org/10.1016/j.ins.2020.03.041.

[57] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Comput. Ind. Eng.*, 149:106854, 2020. doi: 10.1016/j.cie.2020.106854. URL https://doi.org/10.1016/j.cie.2020.106854.

[58] Ping Li, Jin Li, Zhengan Huang, Tong Li, Chong-Zhi Gao, Siu-Ming Yiu, and Kai Chen. Multi-key privacy-preserving deep learning in cloud computing. *Future Gener. Comput. Syst.*, 74:76–85, 2017. doi: 10.1016/j.future.2017.02.006. URL https://doi.org/10.1016/j.future.2017.02.006.

[59] Rongxing Lu, Kevin Heung, Arash Habibi Lashkari, and Ali A Ghorbani. A lightweight privacy-preserving data aggregation scheme for fog computing-enhanced iot. *IEEE Access*, 5:3302–3312, 2017.

[60] Lingjuan Lyu. Lightweight crypto-assisted distributed differential privacy for privacy-preserving distributed learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[61] Xu Ma, Fangguo Zhang, Xiaofeng Chen, and Jian Shen. Privacy preserving multi-party computation delegation for deep learning in cloud computing. *Inf. Sci.*, 459:103–116, 2018. doi: 10.1016/j.ins.2018.05.005. URL https://doi.org/10.1016/j.ins.2018.05.005.

[62] Zhuoran Ma, Jianfeng Ma, Yinbin Miao, and Ximeng Liu. Privacy-preserving and high-accurate outsourced disease predictor on random forest. *Inf. Sci.*, 496:225–241, 2019. doi: 10.1016/j.ins.2019.05.025. URL https://doi.org/10.1016/j.ins.2019.05.025.

[63] Robert Martin, John Demme, and Simha Sethumadhavan. Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. In *39th International Symposium on Computer Architecture (ISCA 2012), June 9-13, 2012, Portland, OR, USA*, pages 118–129. IEEE Computer Society, 2012. doi: 10.1109/ISCA.2012.6237011. URL https://doi.org/10.1109/ISCA.2012.6237011.

[64] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Xiaojin (Jerry) Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017. URL http://proceedings.mlr.press/v54/mcmahan17a.html.

[65] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *CoRR*, abs/1602.05629, 2016. URL http://arxiv.org/abs/1602.05629.

[66] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private language models without losing accuracy. *CoRR*, abs/1710.06963, 2017. URL http://arxiv.org/abs/1710.06963.

[67] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private language models without losing accuracy. *CoRR*, abs/1710.06963, 2017. URL http://arxiv.org/abs/1710.06963.

[68] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 691–706. IEEE, 2019. doi: 10.1109/SP.2019.00029. URL https://doi.org/10.1109/SP.2019.00029.

[69] Ralph C Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[70] Fatemehsadat Mireshghallah, Mohammadkazem Taram, Praneeth Vepakomma, Abhishek Singh, Ramesh Raskar, and Hadi Esmaeilzadeh. Privacy in deep learning: A survey. *CoRR*, abs/2004.12254, 2020. URL https://arxiv.org/abs/2004.12254.

[71] Mahnush Movahedi, Jared Saia, and Mahdi Zamani. Shuffle to baffle: Towards scalable protocols for secure multi-party shuffling. In *35th IEEE International Conference on Distributed Computing Systems, ICDCS 2015, Columbus, OH, USA, June 29 - July 2, 2015*, pages 800–801. IEEE Computer Society, 2015. doi: 10.1109/ICDCS.2015.116. URL https://doi.org/10.1109/ICDCS.2015.116.

[72] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. Toward robustness and privacy in federated learning: Experimenting with local and central differential privacy. *CoRR*, abs/2009.03561, 2020. URL https://arxiv.org/abs/2009.03561.

[73] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001*, pages 116–125. ACM, 2001. doi: 10.1145/501983.502000. URL https://doi.org/10.1145/501983.502000.

[74] Transaction Monitoring Netherlands. What is tmnl?, 2021. URL https://tmnl.nl/summary-eng/. Last Accessed: June 18, 2021.

[75] Monique Ogburn, Claude Turner, and Pushkar Dahal. Homomorphic encryption. In Cihan H. Dagli, editor, *Proceedings of the Complex Adaptive Systems 2013 Conference, Baltimore Marriott Inner Harbor at Camden Yards, Baltimore, Maryland, USA, November 13-15, 2013*, volume 20 of *Procedia Computer Science*, pages 502–509. Elsevier, 2013. doi: 10.1016/j.procs.2013.09.310. URL https://doi.org/10.1016/j.procs.2013.09.310.

[76] Manas A. Pathak, Shantanu Rane, and Bhiksha Raj. Multiparty differential privacy via aggregation of locally trained classifiers. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pages 1876–1884. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper/2010/hash/0d0fd7c6e093f7b804fa0150b875b868-Abstract.html.

[77] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-preserving deep learning: Revisited and enhanced. In Lynn Batten, Dong Seong Kim, Xuyun Zhang, and Gang Li, editors, *Applications and Techniques in Information Security - 8th International Conference, ATIS 2017, Auckland, New Zealand, July 6-7, 2017, Proceedings*, volume 719 of *Communications in Computer and Information Science*, pages 100–110. Springer, 2017. doi: 10.1007/978-981-10-5421-1_9. URL https://doi.org/10.1007/978-981-10-5421-1_9.

[78] Andrea Dal Pozzolo, Olivier Caelen, Yann-Aël Le Borgne, Serge Waterschoot, and Gianluca Bontempi. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Syst. Appl.*, 41(10):4915–4928, 2014. doi: 10.1016/j.eswa.2014.02.026. URL https://doi.org/10.1016/j.eswa.2014.02.026.

[79] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection: A realistic modeling and a novel learning strategy. *IEEE Trans. Neural Networks Learn. Syst.*, 29(8):3784–3797, 2018. doi: 10.1109/TNNLS.2017.2736643. URL https://doi.org/10.1109/TNNLS.2017.2736643.

[80] Bart Preneel. Cryptographic hash functions. *Eur. Trans. Telecommun.*, 5(4):431–448, 1994. doi: 10.1002/ett.4460050406. URL https://doi.org/10.1002/ett.4460050406.

[81] Privacy International. Data breach, 2020.
Available at: https://privacyinternational.org/examples/data-breach, Last Accessed: June 18, 2021.

[82] Anastasia Pustozerova and Rudolf Mayer. Information leaks in federated learning, 2020.

[83] Rabobank. Technology trend report 2020, 2020. URL https://www.rabobank.com/en/about-rabobank/innovation/tech-trends/articles/20200107-technology-trend-report-2020.html. Last Accessed: June 18, 2021.

[84] Md. Atiqur Rahman, Tanzila Rahman, Robert Laganière, and Noman Mohammed. Membership inference attack against differentially private deep learning model. *Trans. Data Priv.*, 11 (1):61–79, 2018. URL http://www.tdp.cat/issues16/tdp.a289a17.pdf.

[85] Jian Ren and Jie Wu. Survey on anonymous communications in computer networks. *Comput. Commun.*, 33(4):420–431, 2010. doi: 10.1016/j.comcom.2009.11.009. URL https://doi.org/10.1016/j.comcom.2009.11.009.

[86] Maria Rigaki and Sebastian Garcia. A survey of privacy attacks in machine learning. *CoRR*, abs/2007.07646, 2020. URL https://arxiv.org/abs/2007.07646.

[87] Paul Sajda. Machine learning for detection and diagnosis of disease. *Annu. Rev. Biomed. Eng.*, 8:537–565, 2006.

[88] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. URL https://www.ndss-symposium.org/ndss-paper/ml-leaks-model-and-data-independent-membership-inference-attacks-and-defenses-on-machine-learning-models/.

[89] Sebastian Schinzel. An efficient mitigation method for timing side channels on the web. In *2nd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, pages 1–6, 2011.

[90] Amazon Web Service. Amazon machine learning: Developer guide, 2021. URL https://docs.aws.amazon.com/machine-learning/latest/dg/what-is-amazon-machine-learning.html.

[91] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. of Symp. Math. Soc., 1971*, volume 20, pages 41–440, 1971.

[92] Micah J. Sheller, G. Anthony Reina, Brandon Edwards, Jason Martin, and Spyridon Bakas. Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation. In Alessandro Crimi, Spyridon Bakas, Hugo J. Kuijf, Farahani Keyvan, Mauricio Reyes, and Theo van Walsum, editors, *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries - 4th International Workshop, BrainLes 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16, 2018, Revised Selected Papers, Part I*, volume 11383 of *Lecture Notes in Computer Science*, pages 92–104. Springer, 2018. doi: 10.1007/978-3-030-11723-8_9. URL https://doi.org/10.1007/978-3-030-11723-8_9.

[93] Micah J Sheller, Brandon Edwards, G Anthony Reina, Jason Martin, Sarthak Pati, Aikaterini Kotrotsou, Mikhail Milchenko, Weilin Xu, Daniel Marcus, Rivka R Colen, et al. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. *Scientific reports*, 10(1):1–12, 2020.

[94] Elaine Shi, TH Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Proc. NDSS*, volume 2, pages 1–17. Citeseer, 2011.

[95] Zhiguo Shi, Ruixue Sun, Rongxing Lu, Le Chen, Jiming Chen, and Xuemin Sherman Shen. Diverse grouping-based aggregation protocol with error detection for smart grid communications. *IEEE Transactions on Smart Grid*, 6(6):2856–2868, 2015.

[96] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1310–1321. ACM, 2015. doi: 10.1145/2810103.2813687. URL https://doi.org/10.1145/2810103.2813687.

[97] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 3–18. IEEE Computer Society, 2017. doi: 10.1109/SP.2017.41. URL https://doi.org/10.1109/SP.2017.41.

[98] Singapore Parliament. Personal data protection act overview, 2014. Available at: pdpc.gov.sg/Overview-of-PDPA/The-Legislation/Personal-Data-Protection-Act, Last Accessed: June 18, 2021.

[99] Nigel P. Smart. *Cryptography Made Simple*. Information Security and Cryptography. Springer, 2016. ISBN 978-3-319-21935-6. doi: 10.1007/978-3-319-21936-3. URL https://doi.org/10.1007/978-3-319-21936-3.

[100] Mengkai Song, Zhibo Wang, Zhifei Zhang, Yang Song, Qian Wang, Ju Ren, and Hairong Qi. Analyzing user-level privacy attack against federated learning. *IEEE J. Sel. Areas Commun.*, 38(10):2430–2444, 2020. doi: 10.1109/JSAC.2020.3000372. URL https://doi.org/10.1109/JSAC.2020.3000372.

[101] Chris Studholme and Ian F. Blake. Multiparty computation to generate secret permutations. *IACR Cryptol. ePrint Arch.*, 2007:353, 2007. URL http://eprint.iacr.org/2007/353.

[102] Latanya Sweeney and Michael Shamos. A multiparty computation for randomly ordering players and making random selections, 2004.

[103] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, and Wenqi Wei. Towards demystifying membership inference attacks. *CoRR*, abs/1807.09173, 2018. URL http://arxiv.org/abs/1807.09173.

[104] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning - (extended abstract). *Inform. Spektrum*, 42(5):356–357, 2019. doi: 10.1007/s00287-019-01205-x. URL https://doi.org/10.1007/s00287-019-01205-x.

[105] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Wenqi Wei, and Lei Yu. Effects of differential privacy and data skewness on membership inference vulnerability. In *First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications, TPS-ISA 2019, Los Angeles, CA, USA, December 12-14, 2019*, pages 82–91. IEEE, 2019. doi: 10.1109/TPS-ISA48467.2019.00019. URL https://doi.org/10.1109/TPS-ISA48467.2019.00019.

[106] Lars Van De Kamp, Chibuike Ugwuoke, and Zekeriya Erkin. Economy: Ensemble collaborative learning using masking. In *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, pages 1–6. IEEE, 2019.

[107] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[108] Wenye Wang and Zhuo Lu. Cyber security in the smart grid: Survey and challenges. *Computer networks*, 57(5):1344–1371, 2013.

[109] Yang Wang, Stephen C. Adams, Peter A. Beling, Steven Greenspan, Sridhar Rajagopalan, Maria C. Velez-Rojas, Serge Mankovski, Steven M. Boker, and Donald E. Brown. Privacy preserving distributed deep learning and its application in credit card fraud detection. In *17th*

IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1-3, 2018, pages 1070–1078. IEEE, 2018. doi: 10.1109/TrustCom/BigDataSE.2018.00150. URL https://doi.org/10.1109/TrustCom/BigDataSE.2018.00150.

[110] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019*, pages 2512–2520. IEEE, 2019. doi: 10.1109/INFOCOM.2019.8737416. URL https://doi.org/10.1109/INFOCOM.2019.8737416.

[111] Kangning Wei, Jinghua Huang, and Shaohong Fu. A survey of e-commerce recommender systems. In *2007 international conference on service systems and service management*, pages 1–5. IEEE, 2007.

[112] Tzer-jen Wei. Secure and practical constant round mental poker. *Inf. Sci.*, 273:352–386, 2014. doi: 10.1016/j.ins.2014.02.151. URL https://doi.org/10.1016/j.ins.2014.02.151.

[113] Mengmeng Yang, Lingjuan Lyu, Jun Zhao, Tianqing Zhu, and Kwok-Yan Lam. Local differential privacy and its applications: A comprehensive survey. *CoRR*, abs/2008.03686, 2020. URL https://arxiv.org/abs/2008.03686.

[114] Wensi Yang, Yuhang Zhang, Kejiang Ye, Li Li, and Cheng-Zhong Xu. FFD: A federated learning based method for credit card fraud detection. In Keke Chen, Sangeetha Seshadri, and Liang-Jie Zhang, editors, *Big Data - BigData 2019 - 8th International Congress, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25-30, 2019, Proceedings*, volume 11514 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2019. doi: 10.1007/978-3-030-23551-2_2. URL https://doi.org/10.1007/978-3-030-23551-2_2.

[115] Xuechao Yang, Xun Yi, Surya Nepal, Andrei Kelarev, and Fengling Han. A secure verifiable ranked choice online voting system based on homomorphic encryption. *IEEE Access*, 6: 20506–20519, 2018. doi: 10.1109/ACCESS.2018.2817518. URL https://doi.org/10.1109/ACCESS.2018.2817518.

[116] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[117] Lingchen Zhao, Qian Wang, Qin Zou, Yan Zhang, and Yanjiao Chen. Privacy-preserving collaborative deep learning with unreliable participants. *IEEE Transactions on Information Forensics and Security*, 15:1486–1500, 2019.

[118] Wenting Zheng, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 724–738. IEEE, 2019. doi: 10.1109/SP.2019.00045. URL https://doi.org/10.1109/SP.2019.00045.

[119] Ligeng Zhu and Song Han. Deep leakage from gradients. In Qiang Yang, Lixin Fan, and Han Yu, editors, *Federated Learning - Privacy and Incentive*, volume 12500 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2020. doi: 10.1007/978-3-030-63076-8_2. URL https://doi.org/10.1007/978-3-030-63076-8_2.